

Kyung-Yong Chwa
J. Ian Munro (Eds.)

LNCS 3106

Computing and Combinatorics

10th Annual International Conference, COCOON 2004
Jeju Island, Korea, August 2004
Proceedings

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

University of Dortmund, Germany

Madhu Sudan

Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos

New York University, NY, USA

Doug Tygar

University of California, Berkeley, CA, USA

Moshe Y. Vardi

Rice University, Houston, TX, USA

Gerhard Weikum

Max-Planck Institute of Computer Science, Saarbruecken, Germany

Kyung-Yong Chwa J. Ian Munro (Eds.)

Computing and Combinatorics

10th Annual International Conference, COCOON 2004
Jeju Island, Korea, August 17-20, 2004
Proceedings

Volume Editors

Kyung-Yong Chwa

Korea Advanced Institute of Science and Technology

Department of Electrical Engineering & Computer Science

373-1 Guseong-dong Yuseong-gu, Daejeon 305-701, Republic of Korea

E-mail: kychwa@jupiter.kaist.ac.kr

J. Ian Munro

University of Waterloo, School of Computer Science

200 University Avenue West, Waterloo, ON, Canada N2L 3G1

E-mail: imunro@uwaterloo.ca

Library of Congress Control Number: Applied for

CR Subject Classification (1998): F.2, G.2.1-2, I.3.5, C.2.3-4, E.1, E.5, E.4

ISSN 0302-9743

ISBN 3-540-22856-X Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media

springeronline.com

© Springer-Verlag Berlin Heidelberg 2004

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik

Printed on acid-free paper SPIN: 11304272 06/3142 5 4 3 2 1 0

Preface

The papers in this volume were selected for presentation at the 10th International Computing and Combinatorics Conference (COCOON 2004), held on August 17–20, 2004 in Jeju Island, Korea. Previous meetings were held in Xi'an (1995), Hong Kong (1996), Shanghai (1997), Taipei (1998), Tokyo (1999), Sydney (2000), Guilin (2001), Singapore (2002), and Big Sky (2003).

In response to the call for papers, 109 extended abstracts were submitted from 23 countries, of which 46 were accepted. The submitted papers were from Belgium (1), Canada (5), China (6), France (1), Germany (6), Hong Kong (8), India (6), Iran (1), Ireland (1), Israel (4), Italy (2), Japan (17), Korea (23), Mexico (3), New Zealand (1), Poland (1), Russia (1), Singapore (5), Sweden (2), Switzerland (3), Taiwan (2), the UK (1), and the USA (9).

Each paper was evaluated by at least three program committee members, with the assistance of referees, as indicated by the referee list found in these proceedings. There were many more acceptable papers than there was space available in the conference schedule, and the program committee's task was extremely difficult. In addition to selected papers, the conference also included three invited presentations by Lars Arge, Jeong Han Kim, and Kokichi Sugihara.

We thank all program committee members and their referees for their excellent work, especially given the demanding time constraints; they gave the conference its distinctive character. We thank all who submitted papers for consideration: they all contributed to the high quality of the conference.

Finally, we thank all the people who worked hard to put in place the logistical arrangements of the conference — our colleagues and our graduate students. It is their hard work that made the conference possible and enjoyable.

June 2004

Kyung-Yong Chwa
J. Ian Munro

Conference Organization

Conference Chair

Kyung-Yong Chwa (KAIST, Korea)

Program Committee Co-Chairs

Kyung-Yong Chwa (KAIST, Korea)

J. Ian Munro (University of Waterloo, Canada)

Program Committee

Nina Amenta (UC Davis, USA)

Michael A. Bender (SUNY at Stony Brook, USA)

Andrej Brodnik (University of Primorska, Slovenia)

Jin-Yi Cai (University of Wisconsin, Madison, USA)

Jik Hyun Chang (Sogang University, Korea)

Danny Z. Chen (University of Notre Dame, USA)

Otfried Cheong (TU Eindhoven, Netherlands)

Francis Y.L. Chin (University of Hong Kong, Hong Kong)

Oscar H. Ibarra (UC Santa Barbara, USA)

Hiroshi Imai (University of Tokyo, Japan)

Tao Jiang (UC Riverside, USA)

Sam Kim (Kyungpook National University, Korea)

Alejandro López-Ortiz (University of Waterloo, Canada)

Joseph S.B. Mitchell (SUNY at Stony Brook, USA)

Rajeev Motwani (Stanford University, USA)

Kunsoo Park (Seoul National University, Korea)

Frank Ruskey (University of Victoria, Canada)

Takeshi Tokuyama (Tohoku University, Japan)

Vijay V. Vazirani (Georgia Institute of Technology, USA)

Peter Widmayer (ETH Zürich, Switzerland)

Chee Yap (New York University, USA)

Organizing Committee

Hwan-Gue Cho (Pusan National University, Korea)

Hee-Chul Kim (Hankuk University of Foreign Studies, Korea)

Sang-Ho Lee (Ewha Womans University, Korea)

Chong-Dae Park (KAIST, Korea)

Chan-Su Shin (Hankuk University of Foreign Studies, Korea)

Sponsors

Korea Advanced Institute of Science and Technology
Korea Science and Engineering Foundation
Korea Information Science Society

Referees

Gagan Aggarwal	Wendy Myrvold
Hee-Kap Ahn	Shubha Nabar
Hans Bodlaender	Mitsunori Ogihara
Michael Brudno	Michael Palis
Cristian Calude	Panos Pardalos
Ke Chen	Chong-Dae Park
Stirling Chow	Heejin Park
Yoojin Chung	Jung-Heum Park
Zhe Dang	Sang-Min Park
John Ellis	Seong-Bae Park
Mihaela Enachescu	C.K. Poon
Satoshi Fujita	Derek Philips
Prasanna Ganesan	Zia Rahman
Chris Gaspar	Srinivasa Rao
Mordecai Golin	Bala Ravikumar
Tero Harju	Sartaj Sahni
Sariel Har-Peled	Kai Salomaa
Seok Hee Hong	Joe Sawada
Jing Huang	Chan-Su Shin
Louis Ibarra	Ulrike Stege
Joo-Won Jung	Doug Stinson
Yosuke Kikuchi	Sergei Vassilvitskii
Sung-Ryul Kim	Mark Weston
Yoonjeong Kim	Xiaodong Wu
Rolf Klein	Jinhui Xu
Christian Knauer	Ying Xu
Hirotada Kobayashi	Sheng Yu
Marc van Kreveld	Xiao Zhou
Henry Leung	An Zhu
Nina Mishra	Eugene Zima

Table of Contents

Invited Presentations

External Geometric Data Structures	1
<i>Lars Arge</i>	
The Poisson Cloning Model for Random Graphs, Random Directed Graphs and Random k -SAT Problems	2
<i>Jeong Han Kim</i>	
Robust Geometric Computation Based on Digital Topology	3
<i>Kokichi Sugihara</i>	

Data Structures and Algorithms I

Adjacency of Optimal Regions for Huffman Trees	13
<i>Kensuke Onishi</i>	
A Construction Method for Optimally Universal Hash Families and Its Consequences for the Existence of RBIBDs	23
<i>Philipp Woelfel</i>	
Towards Constructing Optimal Strip Move Sequences	33
<i>Meena Mahajan, Raghavan Rama, and S. Vijayakumar</i>	

Computational Geometry I

Large Triangles in the d -Dimensional Unit-Cube	43
<i>Hanno Lefmann</i>	
Progress on Maximum Weight Triangulation	53
<i>Francis Y.L. Chin, Jianbo Qian, and Cao An Wang</i>	
Coloring Octrees	62
<i>Udo Adamy, Michael Hoffmann, József Solymosi, and Miloš Stojaković</i>	

Games and Combinatorics

Some Open Problems in Decidability of Brick (Labelled Polyomino) Codes	72
<i>Małgorzata Moczurad and Włodzimierz Moczurad</i>	
Q -Ary Ulam-Rényi Game with Weighted Constrained Lies	82
<i>Ferdinando Cicalese, Christian Deppe, and Daniele Mundici</i>	

Necessary and Sufficient Numbers of Cards
for the Transformation Protocol 92
Koichi Koizumi, Takaaki Mizuki, and Takao Nishizeki

Combinatorial Optimization I

On the Selection and Assignment
with Minimum Quantity Commitments 102
Andrew Lim, Fan Wang, and Zhou Xu

Approximation Algorithms for Multicommodity Flow
and Normalized Cut Problems:
Implementations and Experimental Study 112
Ying Du, Danny Z. Chen, and Xiaodong Wu

Transshipment Through Crossdocks with Inventory and Time Windows ... 122
Andrew Lim, Zhaowei Miao, Brian Rodrigues, and Zhou Xu

Graph Algorithms

Approximated Vertex Cover for Graphs with Perfect Matchings 132
Tomokazu Imamura, Kazuo Iwama, and Tatsue Tsukiji

An Approximation Algorithm
for Weighted Weak Vertex Cover Problem in Undirected Graphs 143
Yong Zhang and Hong Zhu

On the Arrangement of Cliques in Chordal Graphs
with Respect to the Cuts 151
L. Sunil Chandran and N.S. Narayanaswamy

The Worst-Case Time Complexity for Generating All Maximal Cliques ... 161
Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi

Automata and Learning Theory

Regular Expressions for Languages over Infinite Alphabets 171
Michael Kaminski and Tony Tan

On the Power of One-Sided Error Quantum Pushdown Automata
with Classical Stack Operations 179
Masaki Nakanishi

Learning DNFs and Circuits Using Teaching Assistants 188
N. Variyam Vinodchandran

On the Complexity of Samples for Learning 198
Joel Ratsaby

Scheduling

New Results on On-Demand Broadcasting with Deadline via Job Scheduling with Cancellation	210
<i>Wun-Tat Chan, Tak-Wah Lam, Hing-Fung Ting, and Prudence W.H. Wong</i>	

Maximization of the Size and the Weight of Schedules of Degradable Intervals	219
<i>Fabien Baille, Evripidis Bampis, and Christian Laforest</i>	

Minimizing Maximum Lateness on Identical Parallel Batch Processing Machines	229
<i>Shuguang Li, Guojun Li, and Shaoqiang Zhang</i>	

Computational Geometry II

Efficient Algorithms for Approximating a Multi-dimensional Voxel Terrain by a Unimodal Terrain	238
<i>Danny Z. Chen, Jinhee Chun, Naoki Katoh, and Takeshi Tokuyama</i>	

Algorithms for Point Set Matching with k -Differences	249
<i>Tatsuya Akutsu</i>	

Approximation Algorithms for Inscribing or Circumscribing an Axially Symmetric Polygon to a Convex Polygon	259
<i>Hee-Kap Ahn, Peter Brass, Otfried Cheong, Hyeon-Suk Na, Chan-Su Shin, and Antoine Vigneron</i>	

Data Structures and Algorithms II

The Traveling Salesman Problem with Few Inner Points	268
<i>Vladimir G. Deĭneko, Michael Hoffmann, Yoshio Okamoto, and Gerhard J. Woeginger</i>	

A Faster Algorithm for the All-Pairs Shortest Path Problem and Its Application	278
<i>Tadao Takaoka</i>	

Algorithms for the On-Line Quota Traveling Salesman Problem	290
<i>Giorgio Ausiello, Marc Demange, Luigi Laura, and Vangelis Paschos</i>	

Graph Drawing

On the Orthogonal Drawing of Outerplanar Graphs	300
<i>Kumiko Nomura, Satoshi Tayu, and Shuichi Ueno</i>	

Canonical Decomposition, Realizer, Schnyder Labeling and Orderly Spanning Trees of Plane Graphs	309
<i>Kazuyuki Miura, Machiko Azuma, and Takao Nishizeki</i>	

New Bounds on the Number of Edges in a k -Map Graph 319
Zhi-Zhong Chen

Combinatorial Optimization II

Dynamic Storage Allocation and On-Line Colouring Interval Graphs 329
N.S. Narayanaswamy

New Approximation Algorithms
for Some Dynamic Storage Allocation Problems 339
Shuai Cheng Li, Hon Wai Leong, and Steven K. Quek

k -Center Problems with Minimum Coverage 349
Andrew Lim, Brian Rodrigues, Fan Wang, and Zhou Xu

Complexity Theory

On the Extensions of Solovay-Reducibility 360
Xizhong Zheng and Robert Rettinger

The Complexity of Counting Solutions to Systems
of Equations over Finite Semigroups 370
Gustav Nordh and Peter Jonsson

Computational Complexity Classification
of Partition under Compaction and Retraction 380
Narayan Vikas

Parallel and Distributed Architectures

One-to-Many Disjoint Path Covers in a Graph with Faulty Elements 392
Jung-Heum Park

Fault-Tolerant Meshes with Constant Degree 402
Toshinori Yamada

Fault Hamiltonicity of Meshes with Two Wraparound Edges 412
*Kyoung-Wook Park, Hyeong-Seok Lim, Jung-Heum Park,
and Hee-Chul Kim*

On the Expected Time
for Herman's Probabilistic Self-stabilizing Algorithm 422
Toshio Nakata

Computational Biology

An Efficient Online Algorithm for Square Detection 432
Ho-Fung Leung, Zeshan Peng, and Hing-Fung Ting

An Efficient Local Alignment Algorithm for Masked Sequences	440
<i>Jin Wook Kim and Kunsoo Park</i>	
Computing Phylogenetic Roots with Bounded Degrees and Errors Is Hard	450
<i>Tatsuie Tsukiji and Zhi-Zhong Chen</i>	
Inferring a Level-1 Phylogenetic Network from a Dense Set of Rooted Triplets	462
<i>Jesper Jansson and Wing-Kin Sung</i>	
Author Index	473

External Geometric Data Structures

(Invited Paper)

Lars Arge*

Department of Computer Science, Duke University, Durham, NC 27708, USA
`large@cs.duke.edu`

Many modern applications store and process datasets much larger than the main memory of even state-of-the-art high-end machines. Thus massive and dynamically changing datasets often need to be stored in space efficient data structures on external storage devices such as disks, and in such cases the Input/Output (or I/O) communication between internal and external memory can become a major performance bottleneck. Many massive dataset applications involve geometric data (for example, points, lines, and polygons) or data that can be interpreted geometrically. Such applications often perform queries that correspond to searching in massive multidimensional geometric databases for objects that satisfy certain spatial constraints. Typical queries include reporting the objects intersecting a query region, reporting the objects containing a query point, and reporting objects near a query point.

While development of practically efficient (and ideally also multi-purpose) external memory data structures (indexes) has always been a main concern in the database community, most data structure research in the algorithms community has focused on worst-case efficient internal memory data structures. Recently however, there has been some cross-fertilization between the two areas. In this talk we discuss some of the recent advances in the development of dynamic and worst-case efficient external memory geometric data structures. We focus on fundamental structures for important one- and two-dimensional range searching related problems, and try to highlight some of the fundamental techniques used to develop such structures. More comprehensive surveys of external data structure results, as well as external memory algorithms, can e.g. be found in [1–3].

References

1. L. Arge. External memory data structures. In J. Abello, P. M. Pardalos, and M. G. C. Resende, editors, *Handbook of Massive Data Sets*, pages 313–358. Kluwer Academic Publishers, 2002.
2. L. Arge. External-memory geometric data structures. In G. S. Brodal and R. Fagerberg, editors, *EEF Summer School on Massive Datasets*. Springer Verlag, 2004. To appear.
3. J. S. Vitter. External memory algorithms and data structures: Dealing with MASSIVE data. *ACM Computing Surveys*, 33(2):209–271, 2001.

* Supported in part by the National Science Foundation through RI grant EIA-9972879, ITR grant EIA-0112849, CAREER grant CCR-9984099, and U.S.-Germany Cooperative Research Program grant INT-0129182.

The Poisson Cloning Model for Random Graphs, Random Directed Graphs and Random k -SAT Problems

Jeong Han Kim

Microsoft Research
One Microsoft Way, Redmond, WA 98052, USA
jehkim@microsoft.com

In the (classical) random graph model $G(n, p)$ with $p = O(1/n)$, the degrees of the vertices are almost i.i.d Poisson random variables with mean $d = pn + O(1/n)$. Though this fact is useful to understand the nature of the model, it has not been possible to fully utilize properties of i.i.d Poisson random variables. For example, the distribution of the number of isolated vertices is very close to the binomial distribution $B(n, de^{-d})$. In a rigorous proof, however, one has to keep saying how close the distribution is and tracking the effect of the small difference, which is not necessary if the degrees are exactly i.i.d Poisson. Since these kinds of small differences occur almost everywhere in the analysis of the random graph, they make rigorous analysis significantly difficult, if not impossible.

As an approach to minimize such non-essential parts of the analysis, we introduce a random graph model, called the Poisson cloning model, in which all degrees are i.i.d Poisson random variables. Similar models may be introduced for random directed graphs and random k -SAT problems. We will first establish theorems saying that the new models are essentially equivalent to the classical models. To demonstrate how useful the new models are, we will completely analyze some of well-known problems such as the k -core problems of the random graph, the strong component problem of the random directed graph and/or the pure literal algorithm of the random k -SAT problem.

This lecture will be self-contained, especially no prior knowledge about the above mentioned problems are required.

Robust Geometric Computation Based on Digital Topology

Kokichi Sugihara

Department of Mathematical Informatics
University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
sugihara@mist.i.u-tokyo.ac.jp

Abstract. This paper presents a new scheme for designing numerically robust geometric algorithms based on topological consistency. The topology-based approach is one of the most successful principle for designing robust algorithms to solve geometric problems, which the author's group has been developed for a long time. This approach generates efficient algorithms because the floating-point arithmetic can be used, but is not a beginners' technique because topological invariants for individual problems are required in the design of the algorithms. The new scheme presented here uses digital topology instead of individual invariants, and hence will change the topology-based approach from a middle-level technique to a beginners' level technique. The basic idea together with its application to wire bundling is presented.

1 Introduction

Naive implementation of geometric algorithms usually generates unrobust software even though the algorithms are translated into computer language correctly. This is mainly because the algorithms are designed on the assumption that numerical computation can be done correctly, while actual computers generate numerical errors in computation. Numerical errors cause misjudgement on topological structures, generate inconsistency, and thus make software to fail.

To overcome this difficulty, many approaches have been proposed. We can classify them into three groups: the exact computation approach [13, 18], the topology-based approach [12, 15], and the others [1, 7].

In exact computation approach, the resolution of the input data is restricted, and computation is done in precision that is high enough to judge the topological structure always correctly [18].

This approach requires high cost in computation, but general because the same strategy can be applied to a wide class of geometric problems. In this sense, we may say that this approach is suitable to beginners.

In the topology-based approach, on the other hand, we do not expect any accuracy in numerical computation; we place higher priority on keeping the consistency of the topological consistency, and use numerical information only when it is consistent with the topological structure [16]. In this approach, we can use floating-point arithmetic, and hence the resulting computer program runs fast.

However, this approach requires a certain insight to individual problems, because we have to extract topologically invariant properties on which the robustness relies. Hence, we may say that this approach is a middle-level technique.

These two approaches are extremes in mutually opposite directions; the first approach uses perfectly reliable computation, while the second approach does not rely on numerical computation at all.

Other approaches to robustness can be classified into the third group in the sense that they are placed between the first two extreme approaches [1, 7]. In these approaches, they assume a certain fixed precision in computation. In each computation, they evaluate the error, and judge whether the result is reliable. If reliable, they use it; otherwise they do something else. In this way, the resulting software has a binary branch of processing at every computation. Moreover, how to construct the branches much depends on individual problems to be solved. Therefore, we may say that these approaches are the professional-level techniques.

In this paper, we concentrate on the second approach, the topology-based approach, and propose a new scheme for changing it from the middle-level technique to a beginners' technique. The basic idea is the use of digital picture approximation. We first solve the problem approximately on a digital picture, next extract the topological structure from the digital picture, and finally use this topological structure in the topology-based approach. Because we can extract a consistent topological structure from the digital picture approximation, the most difficult step in the original topology-based approach is replaced with an almost automatic step, which is easily carried out.

The original topology-based approach is reviewed in Section 2, and the new scheme is presented in Section 3. An example is given in Section 4, and the conclusion in Section 5.

2 Topology-Based Approach

Suppose that we are given an algorithm for solving a geometric problem such as the construction of a convex hull and the construction of a Voronoi diagram. Let us assume that numerical computation contains errors. A naive implementation of the algorithm usually generates unstable software which fails easily.

Fig. 1(a) shows how a conventional algorithm fails. Let $S = \{J_1, J_2, \dots, J_n\}$ be the set of all the predicates that should be checked in the algorithm. Whether those predicates are true or not is judged on the basis of numerical computations. Since numerical computations contain errors, some of the predicates may be judged incorrectly, which in turn generate inconsistency and the algorithm fails.

Numerical errors cannot be avoided in computation, but still we want to avoid inconsistency. To achieve this goal, the topology-based approach tries to change the structure of the algorithm as shown in Fig. 1(b). We first try to find a maximal subset, say S' , of predicates that are independent from each other, where "independent" means that the truth values of any predicates in S' do not affect the truth values of the other predicates in this subset. The other predicates

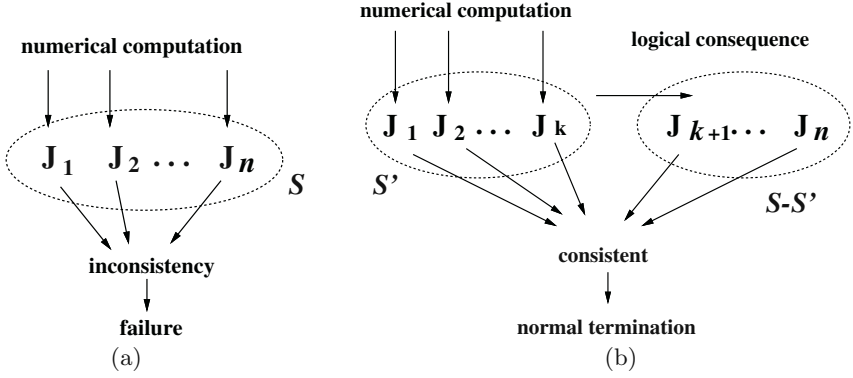


Fig. 1. Basic idea of the topology-based approach.

are dependent in the sense that their truth values are determined as the logical consequence of the truth values of the predicates in S' .

Once we succeed in dividing the predicate set S into a maximal independent subset S' and the other set, we can avoid inconsistency in the following manner. We decide the truth values of the predicates in S' by numerical computation, and adopt the logical consequences of them as the truth values of the other predicates, i.e., the predicates in $S - S'$. Since the predicates in S' are independent, any assignment of the truth values to these predicates does not generate inconsistency. Moreover, since we adopt the logical consequences of these truth values as the truth values of the predicates in $S - S'$, inconsistency never arises.

Note that we cannot guarantee the correctness of the truth values in S' because we have numerical errors. However, once we believe the result of the numerical computation, we can construct a consistent world. This is the basic idea for avoiding inconsistency in the topology-based approach.

The next problem is how to find the maximal independent subset S' of the predicates. This is the most difficult part, and much depends on individual algorithms. We extract purely topological properties of geometric objects, and judge a predicate independent if the topological properties are not violated no matter whether the predicate is true or false. This is the reason why we call this approach the “topology-based approach”.

The following are examples of typical topological properties. In the incremental algorithm for constructing the Voronoi diagram for points in the plane, “the substructure to be deleted by the addition of a new generator should be a tree in the graph theoretic sense” [14, 15]. In the algorithm for cutting a convex polyhedron by a plane, “both of the partitioned vertex-edge graphs of the polyhedron should be connected” [11]. In the incremental algorithm for the Voronoi diagram for line-segments, “the subgraph to be deleted by the addition of a new line-segment should contain a path connecting two Voronoi regions that contain the terminal of the line-segment” [2, 16]. Other examples can be found in the construction of three-dimensional Voronoi diagram [3], the construction of three-dimensional convex hulls [8], and the divide-and-conquer construction of two-dimensional Voronoi diagrams [10].

Though there are many successful cases as stated above, it is still a bottle neck to extract purely topological properties for each algorithm. We will overcome this difficulty in the next section.

3 Use of Digital Topology

The most difficult step in the design of geometric algorithms according to the topology-based approach is to extract purely topological properties that the objects should satisfy. To circumvent this difficulty, we propose a new scheme, in which the topological structure is automatically extracted from a digital-picture approximation. The basic idea is as follows.

As shown in Fig. 2, we first solve the given problem approximately by the digital-picture technique. In the two-dimensional space, the digital picture is an array of small squares called *pixels*. In the three-dimensional space, on the other hand, the digital picture is an array of small cubes called *voxels*. In both cases the space is filled with regular elements, and each element is assigned specific attributes to represent a geometric object approximately.

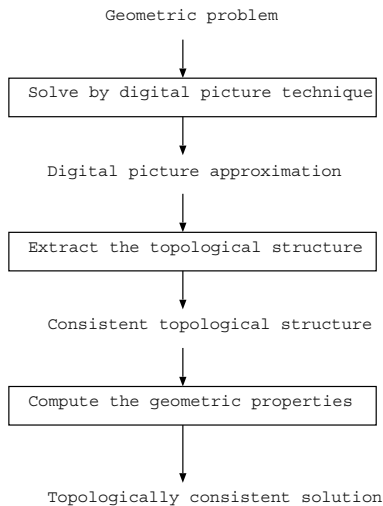


Fig. 2. Geometric computation based on digital topology.

For example, if we want to construct the Voronoi diagram in the plane, we assign to each pixel a specific color corresponding to the nearest generator. This representation is very flexible in the sense that any generalized Voronoi diagram can be constructed in the same manner once we are given a generalized distance between pixels and generators. Indeed for some types of generalized Voronoi diagrams such as the crystal growth Voronoi diagram [6] and the boat-sail Voronoi diagram [9], the digital-picture approximation seems to be the only way for constructing the Voronoi diagram in reasonable time.

Historically, the digital picture approach has not been considered so seriously in computational geometry. This may be mainly because this is a kind of brute-force method, and gives only an approximation of the true resolution; if we want a sufficiently good approximation, we need very high resolution of the pixel/boxel array and hence we need much time in computation.

For our present purpose, however, high resolution is not necessary. What we need is an approximation of the topological structure, and hence only a low resolution approximation is sufficient. Hence, this approximation can be constructed efficiently.

In the second step, we extract the topological structure of the object from the digital picture approximation. For example, if the object is the Voronoi diagram in the plane, we extract the graph structure consisting of Voronoi vertices and Voronoi edges. This can be done automatically, because the Voronoi vertices are points at which three regions meet, and the Voronoi edges are lines at which two regions meet.

By doing so, we can skip extracting invariant topological properties. The space-filling nature of the digital picture automatically tells us a specific example of the topological structure that is consistent in the sense that this topology is actually realized by the example of geometric object represented as the array of pixels or boxels. Therefore, no matter how sophisticatedly the Voronoi diagram is generalized, the common procedure can be applied to get the topological structures.

Once we get the topological structure, we believe this is the correct topological structure just as we do in the topology-based approach. In the third step, we compute the geometric attributes of the obtained topological structure such as the coordinates of the Voronoi vertices and the shapes of the Voronoi edges, and thus generate the final output of the algorithm. Note that this output can be considered as an approximation of the correct solution, and at least the topological consistency is guaranteed because this topology is actually realized by the digital-picture approximation.

Recall that a fatal error of geometric computation comes from topological inconsistency of a geometric object. When we deal with a digital picture, the space-filling structure of pixels/boxels has fixed at the beginning and is never changed. Hence, the topological consistency is kept very easily by representing the geometric object in terms of the digital picture. This way we can avoid inconsistency almost automatically. Thus we can change the topology-based approach from a middle-level technique to a beginners' level technique.

4 Example: Circle Voronoi Diagram and Its Application to Wire Bundling

In the design of modern industrial products such as planes and cars, we often want to estimate the size of a wire bundle for a given set of categories and numbers of electric wires. Fig. 3 shows a section of such a wire bundle.



Fig. 3. Example of a wire bundle.

A simplest way to treat this problem is to reduce it to a two-dimensional disk packing problem: given a set of disks, we want to find the smallest enclosing circle containing all the disks without overlap. To solve this problem strictly is very hard; so we have to be satisfied with a nearly smallest circle.

One successful method for this problem is a “shake-and-shrink” algorithm [17]. In this algorithm, we start with a sufficiently large enclosing circle, and simulate the behavior of the disks when we keep shaking the disks and shrink the enclosing circle step by step. In this algorithm a crucial step is to find empty spaces at which a new disk can be placed without intersecting other disks. This step is repeated a large number of times in the simulation, and the circle Voronoi diagram can reduce the computational time drastically.

Let E be the region bounded by the enclosing circle, and ∂E be its boundary. Let $D = \{d_1, d_2, \dots, d_n\}$ be a set of n disks placed without overlap in E . For any point set X in the plane, let $d(p, X)$ denote the Euclidean distance from point p to X . We define the Voronoi region of d_i ($i = 1, 2, \dots, n$) as

$$R(d_i) = \{p \in E \mid d(p, d_i) < d(p, d_j) \text{ for } j \neq i, d(p, d_i) < d(p, \partial E)\},$$

and the Voronoi region of ∂E as

$$R(\partial E) = \{p \in E \mid d(p, \partial E) < d(p, d_i) \text{ for any } i\}.$$

The region E is partitioned into the Voronoi regions $R(d_1), R(d_2), \dots, R(d_n)$, $R(\partial E)$ and their boundaries. We call this partition the *circle Voronoi diagram* for the disk set D and the enclosing circle ∂E , and denote it by $V(D, \partial E)$. Curves shared by the boundaries of two regions are called *Voronoi edges*, and points shared by the boundaries of three or more regions are called *Voronoi vertices*. Fig. 4 shows an example of the circle Voronoi diagram.

The circle Voronoi diagram contains curved Voronoi edges, and hence is not so easy to compute as the ordinary Voronoi diagram for points. For example, see [4, 5] for an algorithm and implementation.

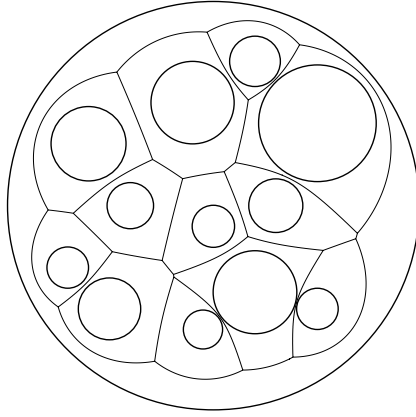


Fig. 4. Circle Voronoi diagram in a circle.

Suppose that we have n nonoverlapping disks d_1, d_2, \dots, d_n and the enclosing circle ∂E , and we want to find all possible placement of the $(n+1)$ -th disk d_{n+1} touching two other disks or the enclosing circle ∂E . If we search for them in a naive manner, it takes $O(n^3)$ time because there are $O(n^2)$ placements of d_{n+1} touching two of the other disks, and for each placement $O(n)$ additional time is required to check whether it intersects other disks.

Using the circle Voronoi diagram, we can decrease the time complexity drastically.

If the disk d_{n+1} touches two other disks, say d_i and d_j , then the center of d_{n+1} is in equal distance from d_i and d_j , which means that the center of d_{n+1} is on the Voronoi edge shared by $R(d_i)$ and $R(d_j)$ or on its extension. The center of d_{n+1} being on the Voronoi edge also means that d_{n+1} does not intersect any other disks, because the Voronoi edge consists of points that are nearer to d_i and d_j than to any other disks.

For any point on a Voronoi edge, let us define the *clearance* as the distance to the two side disks. Let r_{n+1} be the radius of d_{n+1} . The disk d_{n+1} can be placed on a Voronoi edge without intersecting other edges if and only if the clearance at the center of d_{n+1} is equal to or greater than r_{n+1} . Note that the clearance at a point on a Voronoi edge gives its maximal value at one of the terminal points of the edge. Because of these properties, we can find all possible placements of d_{n+1} in the following manner.

Suppose that we already constructed the circle Voronoi diagram $V(D, \partial E)$. Also suppose that for each Voronoi vertex we assigned the clearance. We first select only those Voronoi vertices whose clearance is equal to or greater than r_{n+1} . There are only $O(n)$ vertices, and hence the number of chosen vertices is also $O(n)$ at most. Next, for each of those vertices, we check the incident edges to find the center of d_{n+1} on the edge such that it touches two other disks. Since the total number of Voronoi edges is also bounded by $O(n)$, we can find all possible placements of d_{n+1} in $O(n)$ time.

In this way, the time complexity of finding all possible placements of d_{n+1} touching two other disks and avoiding the interference with other disks can be reduced from $O(n^3)$ to $O(n)$, if we have the circle Voronoi diagram. In this argument, we neglect the time necessary for constructing the Voronoi diagram, but this is reasonable. Indeed, the shake-and-shrink algorithm uses many circle Voronoi diagrams that differs from each other only slightly, and hence the average time complexity for generating one circle Voronoi diagram is negligible.

For this application, we essentially need the Voronoi vertices and their clearances. Hence, the proposed method can be used effectively. Fig. 5 shows an example. Fig. 5(a) represents the disks and the enclosing circle. Fig. 5(b) shows the digital picture representing an approximation of $V(D, E)$. In this digital picture we chose the lowest resolution provided that the Voronoi region of each disk contains at least one pixel, and hence we could construct it very quickly.

From this digital picture, we extracted the topological structure, i.e., the Voronoi vertices and their adjacency relations. Finally, we computed the clearances of the Voronoi vertices, as in Fig. 5(c), where the clearances are shown by

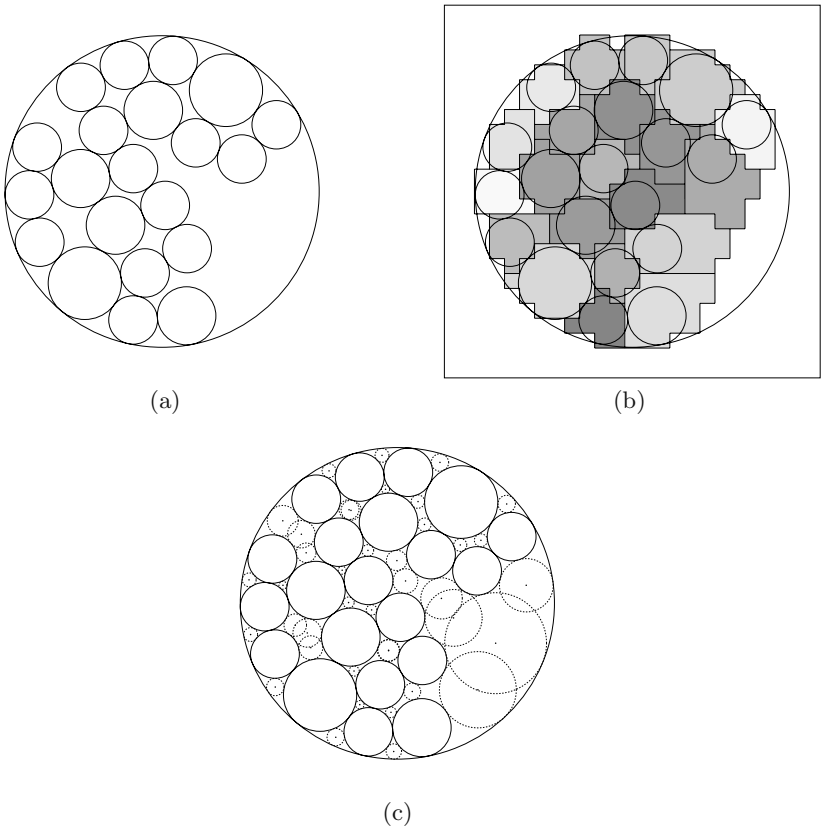


Fig. 5. Digital picture approximation, and the computed vertices and clearances.

locally maximal empty circles represented by broken curves. Using these clearance information, we can find all possible placements of d_{n+1} efficiently.

Note that the extracted topological structure is just an approximation, and hence the clearance value may not represent the distance to the nearest disk; there might be other disks that are nearer. So we have to check also other disks in order to guarantee the non-intersection. However, we found experimentally that it is enough to check the disks whose regions are incident to the Voronoi vertices that are in two or three edge distant from the vertex in consideration. Hence, this intersection check can still be done in constant time.

Using the circle Voronoi diagrams, we could reduce the computation cost of the shake-and-shrink algorithm to a large extent. For example, for a set of 169 wires, the computation time reduced to less than 1% of the time required by the naive method.

5 Concluding Remarks

We presented a new scheme for circumventing the most difficult step of the topology-based approach; the basic idea is to generate a digital-picture approximation of a geometric object and to use its topological structure to guarantee the consistency. The space-filling nature of the digital-picture structure makes it easy for us to attain consistency. Therefore, we can automatically extract the topological structure, and thus can avoid the hard step of finding topologically invariant properties of individual problems and algorithms. In this way, we can change the topology-based approach from a middle-level technique to a beginners' level technique.

Acknowledgments

This work is supported by the 21st Century COE Program on Information Science and Technology Strategic Core, and the Grant-in-Aid for Scientific Research (S) of the Japanese Ministry of Education, Culture, Sports, Science and Technology.

References

1. L. Guibas, D. Salesin and J. Stolfi: Epsilon geometry—Building robust algorithms from imprecise computations. *Proc. 5th ACM Annual Symposium on Computational Geometry*, Saarbrücken, May 1989, pp. 208–217.
2. T. Imai: A topology-oriented algorithm for the Voronoi diagram of polygon. *Proceedings of the 8th Canadian Conference on Computational Geometry*, 1996, pp. 107–112.
3. H. Inagaki, K. Sugihara and N. Sugie: Numerically robust incremental algorithm for constructing three-dimensional Voronoi diagrams. *Proceedings of the 6th Canadian Conference Computational Geometry*, Newfoundland, August 1992, pp. 334–339.

4. D.-S. Kim, D. Kim and K. Sugihara: Voronoi diagram of a circle set from Voronoi diagram of a point set, I. Topology. *Computer Aided Geometric Design*, vol. 18 (2001), pp. 541–562.
5. D.-S. Kim, D. Kim and K. Sugihara: Voronoi diagram of a circle set from Voronoi diagram of a point set, II. Geometry. *Computer Aided Geometric Design*, vol. 18 (2001), pp. 563–585.
6. K. Kobayashi and K. Sugihara: Crystal Voronoi diagram and its applications. *Future Generation Computer Systems*, vol. 18 (2002), pp. 681–692.
7. V. Milenkovic: Verifiable implementations of geometric algorithms using finite precision arithmetic. *Artificial Intelligence*, vol. 37 (1988), pp. 377–401.
8. T. Minakawa and K. Sugihara: Topology-oriented construction of three-dimensional convex hulls. *Optimization Methods and Software*, vol. 10 (1998), pp. 357–371.
9. T. Nishida and K. Sugihara: Approximation of the boat-sail Voronoi diagram and its applications. *Proceedings of the 4th International Conference on Computational Science and Its Applications*, Perugia, May 14–17, 2004 (to appear).
10. Y. Oishi and K. Sugihara: Topology-oriented divide-and-conquer algorithm for Voronoi diagrams. *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing*, vol. 57 (1995), pp. 303–314.
11. K. Sugihara: A robust and consistent algorithm for intersecting convex polyhedra. *Computer Graphics Forum*, EUROGRAPHICS’94, Oslo, 1994, pp. C-45–C-54.
12. K. Sugihara: How to make geometric algorithms robust. *IEICE Transactions on Information and Systems*, vol. E83-D (2000), pp. 447–454.
13. K. Sugihara and M. Iri: A solid modelling system free from topological inconsistency. *Journal of Information Processing*, vol. 12 (1989), pp. 380–393.
14. K. Sugihara and M. Iri: Construction of the Voronoi diagram for “one million” generators in single-precision arithmetic. *Proceedings of the IEEE*, vol. 80 (1992), pp. 1471–1484.
15. K. Sugihara and M. Iri: A robust topology-oriented incremental algorithm for Voronoi diagrams. *International Journal of Computational Geometry and Applications*, vol. 4 (1994), pp. 179–228.
16. K. Sugihara, M. Iri, H. Inagaki and T. Imai: Topology-oriented implementation—An approach to robust geometric algorithms. *Algorithmica*, vol. 27 (2000), pp. 5–20.
17. K. Sugihara, M. Sawai, H. Sano, D.-S. Kim and D. Kim: Disk packing for the estimation of the size of a wire bundle. *Japan Journal of Industrial and Applied Mathematics* (to appear).
18. C. K. Yap: The exact computation paradigm. D.-Z. Du and F. Hwang (eds.): *Computing in Euclidean Geometry, 2nd edition*. World Scientific, Singapore, 1995, pp. 452–492.

Adjacency of Optimal Regions for Huffman Trees

Kensuke Onishi

Graduate School of Information Systems, University of Electro-Communications,
1-5-1 Chofugaoka, Chofu-shi, Tokyo, 182-8585, Japan*

Abstract. The Huffman tree is a binary tree storing each leaf with a key and its weight with the smallest weighted search time (i.e., the weighted sum of external path lengths).

The combinatorial structure of the Huffman tree depends on the weights of keys, and thus the space of weights is tessellated into regions called *optimal regions* associated with the combinatorial structures. In this paper we investigate properties of this tessellation.

We show that each optimal region is convex and non-empty. Moreover, we give a combinatorial necessary and sufficient condition for the adjacency of optimal regions. We also give analogous results for alphabetic trees.

1 Introduction

Huffman tree is a fundamental and popular data structure, and has many applications such as efficient coding of alphabets [2, pp. 402-406].

We consider a binary tree T with n leaves storing keys labeled by $1, 2, \dots, n$. Each key i has a weight $w_i > 0$. We call $\mathbf{w} = \{w_1, w_2, \dots, w_n\}$ the weight vector or the *weight* of T . The *level vector* of T is defined as $\mathbf{l} = (l_1, \dots, l_n)$, where l_i is the level (i.e. depth) of the leaf labeled by i . The cost of T is defined by $\sum_{i=1}^n w_i l_i$. The tree with the minimum cost is called the *Huffman tree* for the set of keys. The tree with the minimum cost under the condition that key i is stored in the i -th leaf from the left for each $i = 1, 2, \dots, n$ is called the *alphabetic tree*. Huffman trees and alphabetic trees are extended binary trees (i.e. trees in which each internal node has two children).

Given a vector $\mathbf{l} = (l_1, \dots, l_n)$ of natural numbers, there exists a binary tree $T(\mathbf{l})$, corresponding to this vector if $\sum_i 2^{-l_i} \leq 1$, and the equality holds if the tree is an extended binary tree. The vector \mathbf{l} is invariant under some permutations of the leaves, and we identify the trees with the same level vector in this paper.

The set \mathcal{L}_n of all n -dimensional level vectors for extended binary trees is defined as

$$\mathcal{L}_n = \left\{ \mathbf{l} \in \mathbb{Z}_{++}^n \mid \sum_{i=1}^n 2^{-l_i} = 1 \right\},$$

* Currently at Department of Mathematical Sciences, School of Science, Tokai University, 1117, Kitakaname, Hiratsuka, Kanagawa, 259-1292, Japan. E-mail: onishi@ss.u-tokai.ac.jp

where $\mathbb{Z}_{++}^n := \{(x_1, \dots, x_n) \mid x_i \in \mathbb{Z}, x_i > 0\}$. For a given weight $\mathbf{w} > \mathbf{0}$, a level vector \mathbf{l} is *optimal* if $\mathbf{l} \cdot \mathbf{w} \leq \mathbf{l}' \cdot \mathbf{w}$ for any other $\mathbf{l}' \in \mathcal{L}_n$. Naturally, \mathbf{l} is optimal if and only if its corresponding tree is a Huffman tree.

For each \mathbf{l} optimal region $R(\mathbf{l})$ is defined as:

$$R(\mathbf{l}) = \{\mathbf{w} \mid \mathbf{l} \cdot \mathbf{w} \leq \mathbf{l}' \cdot \mathbf{w}, \forall \mathbf{l}' \in \mathcal{L}_n\}.$$

Let $\mathbf{W}_n := \{\mathbf{w} = (w_1, \dots, w_n) \mid w_i \in \mathbb{R}, w_i > 0\}$ be the set of all weight vectors, and it called *weight space*. The set of optimal regions for level vectors in \mathcal{L}_n gives a subdivision of \mathbf{W}_n (see Figure 1(left)).

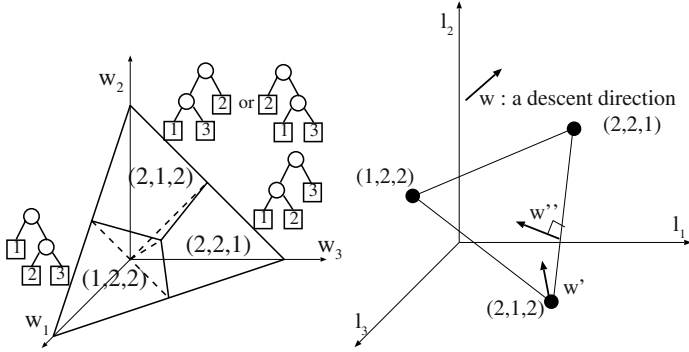


Fig. 1. Optimal regions for Huffman trees with 3 leaves (left) and convex hull for \mathcal{L}_3 (right)

We investigate properties of the optimal regions. Two level vectors \mathbf{l} and \mathbf{l}' are said to be *similar* if there exists a weight \mathbf{w} such that $\mathbf{l} \cdot \mathbf{w} = \mathbf{l}' \cdot \mathbf{w} \leq \mathbf{l}'' \cdot \mathbf{w}, \forall \mathbf{l}'' \in \mathcal{L}_n$. Two level vectors \mathbf{l} and \mathbf{l}' are said to be *adjacent* to each other if there exists a weight \mathbf{w} such that $\mathbf{l} \cdot \mathbf{w} = \mathbf{l}' \cdot \mathbf{w} < \mathbf{l}'' \cdot \mathbf{w}, \forall \mathbf{l}'' \in \mathcal{L}_n$.

Clearly, adjacency implies similarity, although its converse is not always true. Geometrically, similarity means that the corresponding optimal regions share a point on their boundary, whereas adjacency means that they share a facet.

Regarding each level vector as a point in \mathbb{R}^n , we can reduce the computation of Huffman tree for a weight \mathbf{w} to the following optimization problem:

$$\min_{\mathbf{l} \in \mathcal{L}_n} \mathbf{w} \cdot \mathbf{l}.$$

The weight vector gives a descent direction of the level vector. In Figure 1(right) when the direction is \mathbf{w}' , the level vector $(2, 1, 2)$ is optimal. Thus, in the space of level vectors, the adjacency of optimal regions can be interpreted as follows: two level vectors \mathbf{l} and \mathbf{l}' are adjacent to each other if and only if they are connected by an edge on the convex hull of \mathcal{L}_n . In Figure 1, optimal regions $R((2, 1, 2))$ and $R((2, 2, 1))$ share a facet (left), while $(2, 1, 2)$ and $(2, 2, 1)$ are connected by an edge of convex hull of \mathcal{L}_3 (right).

Consider a subset \mathcal{A}_n of \mathcal{L}_n , defined by

$$\mathcal{A}_n = \left\{ \mathbf{l} \in \mathcal{L}_n \left| \begin{array}{l} \text{The labels of leaves } T(\mathbf{l}) \text{ appear in ascending order} \\ \text{from left to right if we traverse the tree in inorder.} \end{array} \right. \right\}.$$

\mathcal{A}_n is a set of all alphabetic trees with n leaves and the correspondence between \mathbf{l} and $T(\mathbf{l})$ is one-to-one. For example, level vector $(1, 2, 2)$ is in \mathcal{A}_3 , while $(2, 1, 2)$ is not (see Figure 1(left)).

The rest of paper is organized as follows: in Section 2, we describe well-known result for Huffman tree. In Section 3, we give our main results: convexity and non-emptiness of an optimal region, and combinatorial conditions for adjacency.

2 Preliminaries

In [4, p.25] the following observation is given.

Lemma A A weighted extended binary tree on a sequence of weights w_1, w_2, \dots, w_n is optimal only if the following conditions hold at any level l :

- 1) for any three nodes a, b, c in level l $w(c) \leq w(a) + w(b)$;
- 2) the weight of any node at level l is greater than or equal to the weight of any node at level $l + 1$.

That is, we can check the optimality by computing maximum and two minimum weights at every level. Further, these conditions are necessary condition. Hence, any weight satisfying these conditions is contained in the optimal region of the tree.

3 Properties of Optimal Regions

In this section we describe properties of optimal regions. We show that optimal region of Huffman or alphabetic tree is convex and non-empty. We also give a combinatorial necessary and sufficient condition for the *adjacency*.

3.1 Convexity and Non-emptiness

First, we show that any optimal region is non-empty. For any $\mathbf{l} \in \mathcal{L}_n$, let $\mathbf{w}(\mathbf{l})$ be the weight vector such that its i -th element is $w_i = 2^{-l_i}$. We show that $T(\mathbf{l})$ becomes the optimal tree for $\mathbf{w}(\mathbf{l})$.

Consider the binary tree $T(\mathbf{l})$. It is shown that the $T(\mathbf{l})$ with $\mathbf{w}(\mathbf{l})$ satisfies the condition of optimality in **Lemma A**. In the lowest L -th level there are only leaves and their weights are equal to 2^{-L} . In the next $(L - 1)$ -th level, internal nodes exist and their weights are the sum of the weights of their child nodes, i.e., $2^{-L} + 2^{-L} = 2^{-(L-1)}$. Leaves in the $(L - 1)$ -th level also have weight $2^{-(L-1)}$. Thus, every node in the $(L - 1)$ -th level has weight $2^{-(L-1)}$. Similarly, it is shown that every node has weight 2^{-l} in the l -th level. Consequently, it is evident that the tree with the weight satisfies the condition 1) and 2) in **Lemma A**.

\mathcal{A}_n is a proper subset of \mathcal{L}_n . If a given vector \mathbf{l} is contained in \mathcal{A}_n and optimal for $\mathbf{w}(\mathbf{l})$ among \mathcal{L}_n , then the tree is also optimal in \mathcal{A}_n .

Consider a weight $\mathbf{w}(\mathbf{l}) + \boldsymbol{\varepsilon}$ where $\boldsymbol{\varepsilon} := (\varepsilon_1, \dots, \varepsilon_n)$ and $|\varepsilon_i| \ll 2^{-l_i}$. It is similarly shown that the level vector \mathbf{l} is optimal for $\mathbf{w}(\mathbf{l}) + \boldsymbol{\varepsilon}$. Thus the optimal region has full dimension.

Moreover, we state that the optimal region is convex. Since the optimal region is defined by intersection of halfspaces, the region is a convex cone.

Theorem 1. *The optimal region of any level vector is convex and has full dimension.*

3.2 Similarity

In this section we give a necessary and sufficient condition for the similarity.

First, we show that there exists a weight such that both of two level vectors \mathbf{l}, \mathbf{l}' are optimal for it if $\mathbf{l} - \mathbf{l}' \in \{-1, 0, 1\}^n$.

Lemma 1. *Let \mathbf{l}, \mathbf{l}' be two n -dimensional level vectors with $\mathbf{l} - \mathbf{l}' \in \{-1, 0, 1\}^n$. Consider a weight $\frac{1}{2}\mathbf{w}(\mathbf{l}) + \frac{1}{2}\mathbf{w}(\mathbf{l}')$. Then, the weight of any node in the l -th level of $T(\mathbf{l})$ and $T(\mathbf{l}')$ is in the range $[\frac{3}{4}2^{-l}, \frac{3}{2}2^{-l}]$.*

Proof. We use induction on the level of $T(\mathbf{l})$. Consider the lowest L -th level in the tree. At this level only leaves exist. The difference of leaf level between $T(\mathbf{l})$ and $T(\mathbf{l}')$ is at most one, the pair of levels is one of the following three types: $(L, L-1)$, (L, L) , $(L, L+1)$, therefore, their weights are either $(2^{-L} + 2^{-(L+1)})/2 = \frac{3}{2} \cdot 2^{-L}$, $(2^{-L} + 2^{-L})/2 = 2^{-L}$ or $(2^{-L} + 2^{-(L-1)})/2 = \frac{3}{4} \cdot 2^{-L}$. Thus, all weights of $\frac{1}{2}\mathbf{w}(\mathbf{l}) + \frac{1}{2}\mathbf{w}(\mathbf{l}')$ are included in $[\frac{3}{4}2^{-L}, \frac{3}{2}2^{-L}]$.

Assume that all weights are included in $[\frac{3}{4}2^{-l}, \frac{3}{2}2^{-l}]$ at the l -th level. We show that weight of any node in the $(l-1)$ -th level is in $[\frac{3}{4} \cdot 2^{-(l-1)}, \frac{3}{2} \cdot 2^{-(l-1)}]$. From the assumption, for any node a in l -th level its weight $w(a)$ satisfies the following inequality: $\frac{3}{4} \cdot 2^{-l} \leq w(a) \leq \frac{3}{2} \cdot 2^{-l}$. Since weight of an internal node is given by the sum of the weights of the two children a and b . The weight of the internal node in the $(l-1)$ -th level is bounded by

$$\frac{3}{4} \cdot 2^{-(l-1)} \leq w(a) + w(b) \leq \frac{3}{2} \cdot 2^{-(l-1)}.$$

Moreover, it is similarly shown that the weight of leaves in the $(l-1)$ -th level is also bounded. Consequently, all nodes in the $(l-1)$ -th level are included in $[\frac{3}{4} \cdot 2^{-(l-1)}, \frac{3}{2} \cdot 2^{-(l-1)}]$. \square

From this lemma, we can check the conditions in **Lemma A**. Since the minimum weight in the l -th level is $\frac{3}{4} \cdot 2^{-l}$ and the maximum weight in the $(l+1)$ -th level is $\frac{3}{2} \cdot 2^{-(l+1)}$, condition 2) is satisfied for any level. From this lemma, it is also shown that the condition 1) is satisfied. It is enough to show only the most extreme case. Since every weight at the l -th level is included in $[\frac{3}{4} \cdot 2^{-l}, \frac{3}{2} \cdot 2^{-l}]$, the extreme case is one that the smallest weight $w(a) = w(b) = \frac{3}{4} \cdot 2^{-l}$

and the largest weight $w(c) = \frac{3}{2} \cdot 2^{-l}$. The inequality $w(c) \leq w(a) + w(b)$ holds in the case.

In the case of an alphabetic tree, the set \mathcal{A}_n is a subset of \mathcal{L}_n and the level vectors are optimal for $\frac{1}{2}\mathbf{w}(\mathbf{l}) + \frac{1}{2}\mathbf{w}(\mathbf{l}')$ in \mathcal{L}_n . Then \mathbf{l} and \mathbf{l}' are optimal for $\frac{1}{2}\mathbf{w}(\mathbf{l}) + \frac{1}{2}\mathbf{w}(\mathbf{l}')$ in \mathcal{A}_n .

From the discussion above, we show the following lemma.

Lemma 2. *Let \mathbf{l}, \mathbf{l}' be level vectors such that $\mathbf{l} - \mathbf{l}' \in \{-1, 0, 1\}^n$. Then, extended binary trees $T(\mathbf{l})$ and $T(\mathbf{l}')$ are optimal for the weight $\frac{1}{2}\mathbf{w}(\mathbf{l}) + \frac{1}{2}\mathbf{w}(\mathbf{l}')$.*

Similarly, we can show that there exists a weight vector such that two or more level vectors are optimal.

Lemma 3. *Consider m level vectors $\mathbf{l}^k = (l_1^k, \dots, l_n^k)$ ($k = 1, \dots, m$) such that $l^{j_1} - l^{j_2} \in \{-1, 0, 1\}^n$ for any j_1 and j_2 . Let $\mathbf{w}(\mathbf{l}^1, \dots, \mathbf{l}^m)$ be a weight whose i -th component is defined by*

$$w_i = \begin{cases} 2^{-q_i} & (\text{when } l_i^j = q_i \text{ for any } j) \\ \frac{1}{2} (2^{-q_i} + 2^{-(q_i+1)}) & (\text{when } l_i^{j_1} = q_i \text{ and } l_i^{j_2} = q_i + 1) \end{cases},$$

where q_i is a positive integer. Any \mathbf{l}^k is optimal for the weight $\mathbf{w}(\mathbf{l}^1, \dots, \mathbf{l}^m)$ ($k = 1, \dots, m$).

Proof. For each pair l^{j_1} and l^{j_2} , the difference of leaf level is at most one. The values of l_i^j s can be one of two cases: 1) all l_i^j have the same value; 2) the values of l_i^j s are divided into two integers q_i and $q_i + 1$. It is shown that $T(\mathbf{l}^k)$ s with $\mathbf{w}(\mathbf{l}^1, \dots, \mathbf{l}^m)$ are optimal by similar discussion as above. \square

Next, we show the converse of lemma 2.

Lemma 4. *If level vectors \mathbf{l} and \mathbf{l}' are similar, then the following relation holds: $\mathbf{l} - \mathbf{l}' \in \{-1, 0, 1\}^n$.*

Proof. Fix a level vector \mathbf{l} and its Huffman tree $T(\mathbf{l})$, whose optimal region is denoted by $R(\mathbf{l})$.

Assume that $R(\mathbf{l})$ is expressed only by inequalities $(\mathbf{l} - \mathbf{l}')\mathbf{w} \leq 0$ where \mathbf{l}' satisfies $\mathbf{l} - \mathbf{l}' \in \{-1, 0, 1\}^n$. Then, for any weight \mathbf{w} on the boundary of $R(\mathbf{l})$, the following relations hold: $(\mathbf{l} - \mathbf{l}')\mathbf{w} = 0$, $\mathbf{l} - \mathbf{l}' \in \{-1, 0, 1\}^n$.

Since \mathbf{l} and \mathbf{l}' are similar, there exists a weight \mathbf{w} such that \mathbf{w} is on the boundary of $R(\mathbf{l})$ and of $R(\mathbf{l}')$, and $T(\mathbf{l})$ and $T(\mathbf{l}')$ are optimal for the weight \mathbf{w} . Because \mathbf{w} is a point on the boundary of the optimal regions, \mathbf{l}' satisfies $\mathbf{l} - \mathbf{l}' \in \{-1, 0, 1\}^n$.

In the rest of this proof, we show that for any \mathbf{l} $R(\mathbf{l})$ is expressed only by inequalities $(\mathbf{l} - \mathbf{l}')\mathbf{w} \leq 0$ ($\mathbf{l} - \mathbf{l}' \in \{-1, 0, 1\}^n$).

Consider the conditions in **Lemma A**, which are necessary conditions for a Huffman tree:

1. for any three nodes a, b, c in the l -th level, $w(c) \leq w(a) + w(b)$;
2. for any node x in the l -th level and y in the $(l + 1)$ -th level, $w(x) \geq w(y)$.

We show that from each condition above a level vector $\mathbf{l}' \neq \mathbf{l}$ satisfying $\mathbf{l} - \mathbf{l}' \in \{-1, 0, 1\}^n$ is computed.

Let $L(a)$ be the set of leaves of the subtree with root node a . The weight of the internal node a is the sum of the weights of leaves in $L(a)$.

Since $L(a)$, $L(b)$ and $L(c)$ are disjoint, the first condition becomes

$$\sum_{i \in L(c)} w_i \leq \sum_{i \in L(a)} w_i + \sum_{i \in L(b)} w_i. \quad (1)$$

The inequality (1) is expressed as $\mathbf{w} \cdot \mathbf{d} \leq 0$, where each element of the vector \mathbf{d} is 0 or ± 1 . Without loss of generality, we assume nodes a and b are siblings in $T(\mathbf{l})$. Let T' be an extended binary tree which is obtained by exchanging the subtree with root node c and the subtree with root node which is the parent of a and b in $T(\mathbf{l})$ (see Figure 2). This tree T' is different from $T(\mathbf{l})$. Let \mathbf{l}' be the level vector of T' . From inequality (1), we have $(\mathbf{l} - \mathbf{l}')\mathbf{w} \leq 0$, and the level vector \mathbf{l}' satisfies $\mathbf{l} - \mathbf{l}' \in \{-1, 0, 1\}^n$.

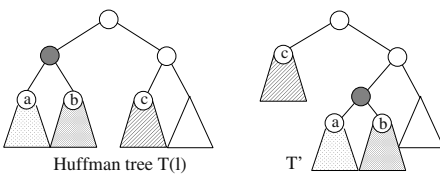


Fig. 2. Huffman tree $T(\mathbf{l})$ and T'

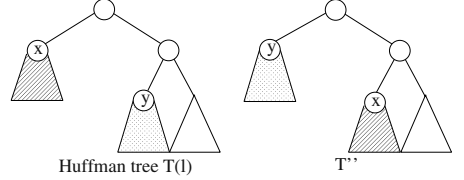


Fig. 3. Huffman tree $T(\mathbf{l})$ and T''

The second condition also gives an inequality:

$$\sum_{i \in L(x)} w_i \geq \sum_{i \in L(y)} w_i. \quad (2)$$

When x is the parent of y , $L(x) \supsetneq L(y)$ holds, otherwise $L(x) \cap L(y) = \emptyset$. Consider the case that x is the parent of y . The inequality (2) becomes $\sum_{i \in L(x) \setminus L(y)} w_i \geq 0$. As every w_i is positive, this inequality gives a trivial bound. Consider another case. For the inequality (2), let T'' be an extended binary tree which is obtained by exchanging the subtree with root x and the subtree with root y in $T(\mathbf{l})$. Let \mathbf{l}'' be the level vector of T'' (see Figure 3). Therefore, inequality (2) is also transformed to $(\mathbf{l} - \mathbf{l}'')\mathbf{w} \leq 0$ and the level vector \mathbf{l}'' satisfies $\mathbf{l} - \mathbf{l}'' \in \{-1, 0, 1\}^n$.

Let $R'(\mathbf{l})$ be a polytope which is determined by these inequalities (1), (2). Since $R'(\mathbf{l})$ is defined by inequalities for $R(\mathbf{l})$, $R'(\mathbf{l})$ contains $R(\mathbf{l})$. On the other hands, the conditions in **Lemma A** are necessary conditions. That is, inequalities (1), (2) are also necessary conditions for optimality of T under \mathbf{w} in \mathbf{W}_n and hence the region $R'(\mathbf{l})$ is subset of $R(\mathbf{l})$. Thus, $R(\mathbf{l})$ is equal to $R'(\mathbf{l})$ as a polytope. The optimal region of \mathbf{l} is defined only by those inequalities (1), (2) whose level vectors satisfy $\mathbf{l} - \mathbf{l}' \in \{-1, 0, 1\}^n$.

Consider alphabetic trees. The conditions of optimal regions of alphabetic trees are similar to those of Huffman trees. The rest of the discussion is analogous. \square

Remark 1. For a level vector \mathbf{l} there may exist several $T(\mathbf{l})$ s. Conversely, when a Huffman tree T is given, its level vector \mathbf{l} is uniquely determined.

Thus, we have the following characterization for the similarity.

Theorem 2. *Two level vectors \mathbf{l} and \mathbf{l}' are similar if and only if $\mathbf{l} - \mathbf{l}' \in \{-1, 0, 1\}^n$.*

Consider all level vectors of alphabetic tree \mathcal{A}_n . In this case, there is a well-known transformation from an alphabetic tree to another, called a *rotation* (see [3]). We show that two alphabetic trees are similar if the trees are transformed to each other by a single rotation.

Corollary 1. *Let \mathbf{l} and \mathbf{l}' be level vectors in \mathcal{A}_n . If alphabetic tree $T(\mathbf{l})$ is transformed to the tree $T(\mathbf{l}')$ by a single rotation, then \mathbf{l} and \mathbf{l}' are similar.*

Proof. Rotation is defined for a pair consisting of a node and its parent. Suppose the node is a right child of the parent. By the rotation the parent and its left subtree move *one* level down, and the node and its right subtree move *one* level up. Other parts are unchanged. Since the difference of level vectors is at most one, therefore, \mathbf{l} and \mathbf{l}' are similar. \square

Remark 2. For binary search tree, similar relation was already shown in [5].

3.3 Adjacency

Two level vectors are *similar* if and only if the difference between the levels of each leaves is at most one. In this section we give a necessary and sufficient conditions that two level vectors are adjacent.

First, we state non-adjacency condition.

Lemma 5. *Consider level vectors \mathbf{l}, \mathbf{l}' . If there are two level vectors \mathbf{m}, \mathbf{m}' such that $\mathbf{l} + \mathbf{l}' = \mathbf{m} + \mathbf{m}'$, then \mathbf{l} and \mathbf{l}' are not adjacent.*

Proof. Consider a weight \mathbf{w} such that $\mathbf{l} \cdot \mathbf{w} = \mathbf{l}' \cdot \mathbf{w}$.

Compute the inner product between $\mathbf{l} + \mathbf{l}'$ and the weight \mathbf{w} :

$$\begin{aligned} (\mathbf{l} + \mathbf{l}') \cdot \mathbf{w} &= (\mathbf{m} + \mathbf{m}') \cdot \mathbf{w} \\ 2\mathbf{l} \cdot \mathbf{w} &= \mathbf{m} \cdot \mathbf{w} + \mathbf{m}' \cdot \mathbf{w} \end{aligned}$$

Since every element of the level vector \mathbf{l} and the weight \mathbf{w} are positive, $\mathbf{l} \cdot \mathbf{w}$, $\mathbf{m} \cdot \mathbf{w}$ and $\mathbf{m}' \cdot \mathbf{w}$ are positive. Thus the following inequalities hold:

$$\mathbf{l} \cdot \mathbf{w} \geq \mathbf{m} \cdot \mathbf{w} \text{ or } \mathbf{l} \cdot \mathbf{w} \geq \mathbf{m}' \cdot \mathbf{w}.$$

When this inequality holds, \mathbf{l} is not optimal for \mathbf{w} . When this equality holds, $\mathbf{l} \cdot \mathbf{w} = \mathbf{m} \cdot \mathbf{w} = \mathbf{m}' \cdot \mathbf{w}$. In this case, $\mathbf{l}' \cdot \mathbf{w}$ has the same value as $\mathbf{l} \cdot \mathbf{w}$. Thus the value of the inner product between the four level vectors and \mathbf{w} is the same. Since \mathbf{m} is also optimal, \mathbf{l} and \mathbf{l}' are not adjacent to each other. \square

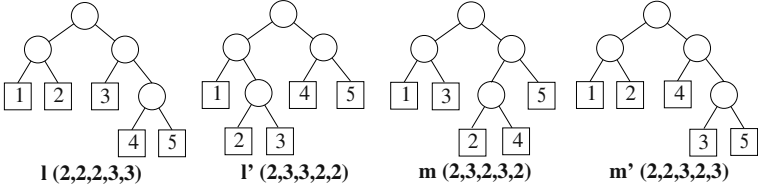


Fig. 4. Similar Huffman trees

Figure 4 shows four Huffman trees. These Huffman trees are simultaneously optimal for $w = (1/4, 3/16, 3/16, 3/16, 3/16)$. In this case the pairs (l, l') , (m, m') are non-adjacent. The other pairs are adjacent. Geometrically, there is a hyperplane whose normal vector is w such that the hyperplane is supporting hyperplane of the convex hull of \mathcal{L}_n and contains these level vectors.

Next, we state the following lemma.

Lemma 6. *If level vectors l, l' are non-adjacent, then the either of the following conditions holds:*

1. *there exists an index i such that $|l_i - l'_i| \geq 2$;*
2. *there exist two level vectors m and m' such that $l + l' = m + m'$;*

where $l_i(l'_i)$ is the i -th element of $l(l')$, respectively.

Proof. Omit. □

Finally, we have a combinatorial necessary and sufficient condition for the adjacency from Lemma 5 and Lemma 6.

Theorem 3. *Two n -dimensional level vectors l and l' are adjacent if and only if the both of the following conditions holds:*

1. $l - l' \in \{-1, 0, 1\}^n$ and
2. *for any other level vectors m and m' , $l + l' \neq m + m'$.*

4 Conclusion

In this paper we deal with properties of optimal regions for Huffman trees and alphabetic trees. These trees are expressed by level vectors. The optimal region of level vector is convex, non-empty and full-dimensionality. We characterize similarity and adjacency between two level vectors: two level vectors are similar if and only if the difference of the level vectors is contained in $\{-1, 0, 1\}^n$. Two level vectors of l, l' are adjacent if and only if the two level vectors are similar and for any other level vectors m, m' it holds that $l + l' \neq m + m'$.

We state two applications of our results: one is computation of *similar* extended binary trees for a given extended binary tree, another is enumeration of all extended binary tree by using similarity and reverse search.

In the former application, when an extended binary tree $\mathbf{l} = (l_1, \dots, l_n)$ is given, similar trees are computed. The similar trees are useful for dynamic Huffman tree, or coding. Consider the case of Huffman tree. In the proof of Lemma 4, we state that *one* inequality corresponds to *one* similar level vector. For example, the similar vector $\mathbf{l}' = (l'_1, \dots, l'_n)$ for inequality (1) becomes as follows:

$$l'_i = \begin{cases} l_i + 1 & (\text{if } i \in L(c)) \\ l_i - 1 & (\text{if } i \in L(a) \cup L(b)) \\ l_i & (\text{otherwise}) \end{cases} .$$

Consider the case of alphabetic tree. Ramanan showed optimality condition for alphabetic tree in [6]. The condition for alphabetic tree is similar to that for Huffman tree. Moreover, we state that *one* inequality of the condition corresponds *one* similar alphabetic tree. Hence, for given alphabetic tree, every similar alphabetic trees can be computed from the inequality. For example, consider a level vector $\mathbf{l} = (1, 3, 3, 2)$. In the tree $T(\mathbf{l})$, node 2 and node 3 are the descendant of the sibling node of node 1, therefore, $w_1 \geq w_2 + w_3$ holds. In this case, node 1 is down by one level, and node 2 and node 3 are up by one, then a similar level vector $(2, 2, 2, 2)$ is computed.

In the latter application, we consider enumeration of all extended binary trees by using *reverse search* [1]. Reverse search is a method of enumeration of nodes of a given graph by using *parent relation*. We only state how to use reverse search in this case. The node set of a graph is \mathcal{L}_n (or \mathcal{A}_n). Two nodes are connected by an edge if the corresponding level vectors are similar. Lexicographic order is induced in \mathcal{L}_n . The *parent relation* is defined as follows: the parent of a given vector \mathbf{l} is the smallest level vector among all similar level vector of \mathbf{l} by the order. The enumeration starts from a node $(1, 2, \dots, n-1, n-1)$ which is the smallest node in \mathcal{L}_n . Some procedures are designed for using reverse search, for instance, computation of the parent and calculation of the next child node. These procedures can be programmed easily in this case.

Moreover the case of alphabetic tree is equivalent to enumerating objects whose number is equal to Catalan number. Such objects have many different combinatorial interpretations. When considering a transformation from the level vector to other interpretations, the objects are also enumerated. One of the interesting interpretation is the triangulations of a convex polygon. In [7], it is shown that a rotation among binary trees is equal to a diagonal flip among the triangulations of convex polygon. The similarity of extended binary trees is a more useful relation than the rotation. Therefore, this relation would be used for optimization on extended binary trees, or on triangulations of a convex polygon.

Finally, we describe the relation between the optimal region of Huffman tree and that of alphabetic tree for *one* level vector. Consider a level vector \mathbf{l} in \mathcal{A}_n . This vector is also included in \mathcal{L}_n . Let $R(\mathbf{l})$ and $S(\mathbf{l})$ be optimal regions in \mathcal{L}_n and in \mathcal{A}_n , respectively. Any point in \mathcal{L}_n is optimal in \mathcal{A}_n , but the converse is not true. Then the following relation holds:

$$R(\mathbf{l}) \subsetneq S(\mathbf{l}).$$

In information theory it is shown that the average length of a code using a Huffman tree becomes optimal and that using an alphabetic tree does not attain to optimal. This fact is consistent with above the relation above.

Acknowledgements

The author would like to thank Prof. Tomomi Matsui, Prof. David Avis and Prof. Luc Devroye for useful discussions and pointing out several errors. The author also thanks Prof. Mamoru Hoshi Prof. Takeshi Tokuyama, Prof. Ian Munro, Prof. Yasuko Matsui for useful suggestions.

References

1. D. Avis: Living with lrs, *Lecture Notes in Computer Science*, Vol. 1763, 2000, pp. 47-56.
2. D.E. Knuth: *The Art of Computer Programming*, Vol. 1, Third Edition, Addison-Wesley, 1997.
3. D.E. Knuth: *The Art of Computer Programming*, Vol. 3, Second Edition, Addison-Wesley, 1998.
4. S. V. Nagaraj: Optimal binary search trees, *Theoretical Computer Science*, no. 188, 1997, pp. 1 - 44.
5. K. Onishi and M. Hoshi: Generation of subdivision for binary search tree with optimal path length of search probability, *IPSJ SIG-Notes Algorithms Abstract*, no. 76, 2001, pp. 65 - 72(*in Japanese*).
6. P. Ramanan: Testing the optimality of alphabetic trees, *Theoretical Computer Science*, no. 93, 1992, pp. 279 - 301.
7. D.D. Sleator, R.E. Tarjan and W.P. Thurston: Rotation distance, triangulations, and hyperbolic geometry, *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, 1986, pp.122 - 135.

A Construction Method for Optimally Universal Hash Families and Its Consequences for the Existence of RBIBDs (Extended Abstract)

Philipp Woelfel

FB Informatik 2, Univ. Dortmund, D-44221 Dortmund, Germany
philipp.woelfel@cs.uni-dortmund.de

Abstract. We introduce a method for constructing optimally universal hash families and equivalently RBIBDs. As a consequence of our construction we obtain minimal optimally universal hash families, if the cardinalities of the universe and the range are powers of the same prime. A corollary of this result is that the necessary condition for the existence of an RBIBD with parameters (v, k, λ) , namely $v \bmod k = \lambda(v - 1) \bmod (k - 1) = 0$, is sufficient, if v and k are powers of the same prime. As an application of our construction, we show that the k -MAXCUT algorithm of Hofmeister and Lefmann [9] can be implemented such that it has a polynomial running time, in the case that the number of vertices and k are powers of the same prime.

1 Introduction and Results

The concept of universal hashing as introduced by Carter and Wegman [4] in 1979 has found a wide variety of applications. Besides of being an important tool for hashing schemes (see e.g. [4, 8, 7]), universal hashing has been used in many other areas of computer science such as complexity theory, cryptography or algorithmics. The importance of this concept has led to a search for practical hash families with good properties and to the investigation of the combinatorial structure of such hash families. In 1994, Stinson [15] has drawn the first connections between universal hash families and combinatorial designs such as resolvable balanced incomplete block designs or orthogonal arrays. Later on, more connections to combinatorial designs and other structures as, e.g., authentication codes were discovered. While on one hand, such connections have led to new applications of universal hashing in cryptography or the theory of combinatorial designs, they also allowed new constructions of universal hash families and new ways of analyzing them (for some references see e.g. [16, 17, 1, 3]).

Definition 1. An $(N; u, r)$ -hash family \mathcal{H} is a family of N functions $U \rightarrow R$, where U and R are finite sets of cardinality $u > 1$ and $r > 1$, resp.

If \mathcal{H} is a hash family of functions $U \rightarrow R$, then U is called *universe* and R is called *range* of \mathcal{H} . We call the elements from the universe *keys*, and say that

two keys $x_1, x_2 \in U$ *collide* under a function $h \in \mathcal{H}$, if $h(x_1) = h(x_2)$. The *collision probability* of x_1 and x_2 is the probability that x_1 and x_2 collide under a randomly chosen function in \mathcal{H} , i.e., $\mathbf{Prob}_{h \in \mathcal{H}}(h(x_1) = h(x_2))$.

It was the original intention behind using hash families in hashing schemes to guarantee that under a randomly chosen hash function the expected number of collisions among a set of keys is low. This motivates the following common definition.

Definition 2 ([4, 15]). *A hash family with universe U is ϵ -universal, if any two distinct keys $x_1, x_2 \in U$ have a collision probability of at most ϵ . A $(1/r)$ -universal $(N; u, r)$ -hash family is simply called universal.*

Many constructions of ϵ -universal hash families have been proposed, and some of them can be implemented by means of simple arithmetic operations such as multiplication [6] or convolution [11]. But besides the search for efficient hash families, there has also been some interest in investigating the properties of hash families with extremely small cardinalities or collision probabilities.

We denote by \gcd and lcm the greatest common divisor and the least common multiple, resp. It can be shown [13] that no ϵ -universal $(N; u, r)$ -hash family exists, if $\epsilon < (u - r)/(r(u - 1))$, and that for any ϵ -universal $(N; u, r)$ -hash family with $\epsilon = (u - r)/(r(u - 1))$, it is $N \geq (u - 1)/\gcd(u - 1, r - 1)$. These properties justify the following definition.

Definition 3 ([13]). *An ϵ -universal $(N; u, r)$ -hash family \mathcal{H} is called optimally universal or short OU, if $\epsilon = \epsilon_{\text{opt}}(u, r) := (u - r)/(r(u - 1))$. If in addition $N = (u - 1)/\gcd(u - 1, r - 1)$, then \mathcal{H} is called minimal optimally universal.*

Sarwate [13] has presented constructions of minimal optimally universal $(N; q^n, q^m)$ -hash families for any prime power q in the cases $m = 1$ and $m = n - 1$ ($n \geq 1$). He states though, that while several ad hoc constructions of small OU universal $(N; q^n, q^m)$ -hash families can be obtained for many values of n and m , there is no general construction method known that produces minimal OU hash families for all n and m . The main purpose of this paper is to present such a general construction method, as summarized by the following theorem.

Theorem 1. *Let q be an arbitrary prime power. For any $1 \leq m \leq n$ there exists a minimal optimally universal $(N; q^n, q^m)$ -hash family.*

The construction is based on the definition of a new type of hash families, called *partitioned universal*. Our proof does not only show the existence of the optimally universal hash families, but describes in fact an efficient algorithm for constructing them. Moreover, the resulting hash functions can be evaluated fairly simple by means of finite field arithmetic. We show below, that our construction has algorithmic consequences as well as consequences for the existence of certain important combinatorial objects.

Definition 4. A balanced incomplete block design (*short: BIBD*) is a pair (V, \mathcal{B}) , where V is a set of points and \mathcal{B} is a set of k -subsets of V called blocks, such that for any two distinct points $p, p' \in V$ there are exactly λ blocks $b \in \mathcal{B}$ where $\{p, p'\} \subseteq b$. The parameters k and λ are called block size and intersection number, resp. If some blocks form a partition of V , then the set of these blocks is called parallel class. A BIBD is resolvable, if its blocks can be partitioned into parallel classes. A resolvable BIBD with v points, block-size k and intersection number λ is denoted by $\text{RBIBD}_\lambda[k; v]$.

RBIBDs are well investigated combinatorial structures and a lot of research has been spent on finding RBIBDs with certain parameters or on disproving their existence (a broad overview on results can be found in the monographs [2, 5]). BIBDs and RBIBDs have also some algorithmic applications. E.g., the construction of layouts for redundant disk arrays in [10, 14] is based on BIBDs and Hofmeister and Lefmann [9] show how to find a large k -cut in a graph with n vertices using an $\text{RBIBD}_\lambda[k; n]$. The authors complain, though, that although various algebraic construction methods for BIBDs are known in the literature, not a lot of attention has been paid to their exact running times.

Besides of the algorithmic problem of constructing RBIBDs, it is an important open question of design theory, for which parameters RBIBDs do exist. It is well-known that the following two conditions are necessary for the existence of an $\text{RBIBD}_\lambda[k; v]$:

$$(C1) \quad v \equiv 0 \pmod{k} \quad \text{and} \quad (C2) \quad \lambda(v-1) \equiv 0 \pmod{k-1}.$$

It is easy to see that the second condition is fulfilled if and only if λ is a multiple of $\lambda_{\min}(k, v) := (k-1)/\gcd(k-1, v-1)$. Thus, $\lambda_{\min}(k, v)$ is the minimal intersection number an RBIBD over v points and with block size k can have.

As Stinson [15] has shown, an OU hash family \mathcal{H} of functions $U \rightarrow R$ describes an RBIBD, by taking U as point set and each set $h^{-1}(y)$ with $y \in R$ and $h \in \mathcal{H}$ as a block. Clearly, for a fixed h , the blocks $h^{-1}(y)$ with $y \in R$ form a parallel class. Taking into account that in an OU hash family any pair of keys has the same collision probability [13], it is easy to see that this block design is in fact an RBIBD. This construction can also be reversed such that one obtains from any RBIBD an OU hash family.

Theorem 2 ([15]). *An optimally universal $(N; u, r)$ -hash family exists if and only if there is an $\text{RBIBD}_\lambda[k; v]$ with $v = u$, $k = u/r$, and $\lambda = N(u-r)/(r(u-1))$.*

Plugging the minimal intersection number λ_{\min} into this theorem, it is easy to see that a minimal optimally universal $(N; u, r)$ -hash family exists if and only if there exists an $\text{RBIBD}_\lambda[u/r; u]$ with $\lambda = \lambda_{\min}(u/r, u)$. Using this equivalence, our construction method of minimal OU hash families from Theorem 1 implies the existence of an $\text{RBIBD}_{\lambda_{\min}(k, v)}[k; v]$ for any $k = q^{n-m}$ and $v = q^n$ where q is a prime power (and $m \leq n$). Note that if $v \neq 0$, then (C1) obviously implies $k \leq v$. Since in addition any λ satisfying (C2) is a multiple of $\lambda_{\min}(k, v)$, for

any such λ an $\text{RBIBD}_\lambda[k; v]$ can be obtained by taking multiple copies of an $\text{RBIBD}_{\lambda_{\min}(k, v)}[k; v]$.

Corollary 1. *The necessary conditions (C1) and (C2) for the existence of an $\text{RBIBD}_\lambda[k; v]$ are sufficient, if k and v are powers of the same prime.*

Due to the fact that the hash family from Theorem 1 can be constructed efficiently, we obtain also an efficient algorithm for constructing these RBIBDs. Therefore, they may be useful for algorithmic problems.

2 Application to k -MAXCUT

In [9], Hofmeister and Lefmann have presented a deterministic algorithm for computing a large k -cut in a graph, using RBIBDs. Let $G = (V, E)$ be an undirected graph whose edges are weighted by a function $w : E \rightarrow \mathbb{N}_0$. By $w(G)$ we denote the sum of edge weights, i.e., $w(G) = \sum_{e \in E} w(e)$. A *balanced k -cut* of G is a partition of the vertices V into k sets V_1, \dots, V_k of equal size. The *cut size* of a k -cut (V_1, \dots, V_k) is the sum of weights of all *crossing* edges, that is edges $e = \{v, w\}$ with $(v, w) \in V_i \times V_j$ with $i \neq j$.

Assume that an $\text{RBIBD}_\lambda[n; k]$ is given by a description of the blocks, and that the list of these blocks is already arranged in such a way, that all blocks of each parallel class appear one after the other. Hofmeister and Lefmann have shown that in this case a balanced k -cut of size at least $w(G) \cdot \frac{k-1}{k} \cdot (1 + \frac{1}{n-1})$ can be computed for any graph G with n vertices in time $O(k(n+m) + \lambda n^2)$. Note that for any constant $0 < \epsilon < 1/(k-1)$, $k \geq 2$, it is NP-complete to decide whether a graph with m edges admits a k -cut of size at least $m \cdot \frac{k-1}{k} \cdot (1 + \epsilon)$ [12].

The algorithm of Hofmeister and Lefmann relies though, on having the explicit description of an RBIBD with appropriate parameters at hand. Since most known RBIBDs are ad-hoc constructions and algorithms for the construction of RBIBDs for a wide variety of parameters are rarely described in literature, it is not clear for which parameters the algorithm can be implemented. In fact, the only statement Hofmeister and Lefmann can make is that for n being a multiple of 3 and n , $n-1$ or $n-2$ being a prime power, a polynomial time algorithm exists for the construction of the desired RBIBDs.

Using our construction of optimally universal hash families, we obtain now explicit implementations of their algorithm for all n and k being powers of the same prime. We assume a RAM model allowing arithmetics over natural numbers and finite fields of order at most n in constant time.

Corollary 2. *If n and k are powers of the same prime, then for any graph with n vertices and m edges a balanced k -cut of weight at least $w(G) \cdot \frac{k-1}{k} \cdot (1 + \frac{1}{n-1})$ can be computed in time $O(k(n+m) + k \cdot n^2 / \gcd(n-1, k-1))$.*

Note that this result is obtained by simply plugging our RBIBD construction into the algorithm of Hofmeister and Lefmann. We can even do better if we want to find k -cuts with $k > \sqrt{n}$ by a very similar and simple algorithm using OU hash families.

Let $G = (V, E)$ be a graph with n vertices and let \mathcal{H} be an ϵ -universal $(N; n, k)$ -hash family with universe V and range $\{1, \dots, k\}$. Any hash function $h \in \mathcal{H}$ defines a cut (V_1, \dots, V_k) on the graph, by letting $V_i = \{v \in V \mid h(v) = i\}$. Moreover, if \mathcal{H} is optimally universal, then the cut is balanced, because it is well-known that in this case $|h^{-1}(i)| = n/k$ (see e.g. [13]). Now choose a random hash function h from \mathcal{H} and consider the cut (V_1, \dots, V_k) defined by h . Since \mathcal{H} is ϵ -universal, each edge is a crossing edge with a probability of at least $1 - \epsilon$. Hence, by the linearity of expectation, the expected cut size is bounded below by $(1 - \epsilon)w(G)$. This means that there exists a hash function $h \in \mathcal{H}$ which defines a cut having at least that size. Using $\epsilon = \epsilon_{\text{opt}}(n, k) = (n - k)/(k(n - 1))$ if \mathcal{H} is optimally universal, yields an edge weight of at least $w(G) \cdot (1 - \frac{n-k}{k(n-1)}) = w(G) \cdot \frac{k-1}{k} \cdot (1 + \frac{1}{n-1})$.

As we will show in the remainder of this paper, the hash functions from the minimal OU hash family \mathcal{H} claimed to exist in Theorem 1 can be enumerated in time $O(N)$, where $N = (n - 1)/\gcd(n - 1, k - 1)$. Moreover, under the assumption of a RAM where (finite field) arithmetics can be done in constant time, the hash functions can be evaluated in constant time. Once we have picked a hash function $h \in \mathcal{H}$, we can evaluate the hash function values of all n vertices in time $O(n)$ in order to compute the corresponding cut (V_1, \dots, V_k) . Now, for each set V_i , the sum of weights of edges within V_i (that is edges $e = \{v, w\}$ with $v, w \in V_i$), can be computed in time $O(n^2/k^2)$ (assuming that the graph is given by an adjacency matrix). This way, we can compute the sum $s(V_1, \dots, V_k)$ of weights of all non-crossing edges in time $O(n^2/k)$ and the k -cut with the minimal value $s(V_1, \dots, V_k)$ has the maximal weight $w(G) - s(V_1, \dots, V_k)$. Hence, the total running time is bounded by $O(N \cdot n^2/k) = O(n^3/(k \cdot \gcd(n - 1, k - 1)))$. For $k > \sqrt{n}$ this is better than the result of Corollary 2.

Theorem 3. *If n and k are powers of the same prime, then for any graph with n vertices given by an adjacency matrix, a balanced k -cut of weight at least $w(G) \cdot \frac{k-1}{k} \cdot (1 + \frac{1}{n-1})$ can be computed in time $O(n^3/(k \cdot \gcd(n - 1, k - 1)))$.*

3 Partitioned Universal Hash Families

In the following, we define a new type of hash families, called partitioned universal. They are the main ingredients for our construction of optimally universal hash families.

Definition 5. *An $(N; u, r)$ -hash family is partitioned ϵ -universal if it is ϵ -universal and if there exists an equivalence relation partitioning the universe in equivalence classes of size r such that any two distinct keys from the same equivalence class have a collision probability of 0. In the case $\epsilon = 1/r$ the hash family is simply called partitioned universal.*

In the remainder of the text we use the convention that if the universe of a partitioned universal hash family with range R is written as $W \times R$, then the

equivalence classes to which the above definition refers to, are the classes U_x , $x \in W$, where $U_x = \{(x, x') \mid x' \in R\}$.

While we have defined partitioned ϵ -universal hash families mainly in order to prove Theorem 1, there are several examples of well-known ϵ -universal hash families, which in fact turn out to be ϵ -partitioned universal. One example where this is the case is the multiplicative $(2/r)$ -universal hash family defined in [6]. As we will see in the following, partitioned universal hash families are quite easy to construct by means of another type of hash families, defined by Stinson in 1996.

Definition 6 ([17]). Let $\mathcal{H} : U \rightarrow R$ be an $(N; u, r)$ -hash family, where R is an additive abelian group. \mathcal{H} is called ϵ - Δ -universal if for any two distinct keys $x_1, x_2 \in U$ and any $d \in R$, $\text{Prob}_{h \in \mathcal{H}}(h(x_1) - h(x_2) = d) \leq \epsilon$. In the case $\epsilon = 1/r$, \mathcal{H} is simply called Δ -universal.

Here is a well-known construction (see e.g. [17]): Denote by \mathbb{F}_q a finite field of order q and let $U = (\mathbb{F}_q)^n$ and $R = (\mathbb{F}_q)^m$ be extension fields of \mathbb{F}_q . If $\phi : U \rightarrow R$ is a surjective homomorphism then it is easy to see that the family $\{f_a \mid a \in \mathbb{F}_{q^n}\}$, where $f_a : U \rightarrow R$, $x \mapsto \phi(ax)$ is Δ -universal. If $n < m$, then any Δ -universal $(q^m; q^m, q^m)$ -hash family is also Δ -universal for an arbitrary q^n -element subset of the universe. Thus, one gets the following result.

Lemma 1 ([17]). Let q be a prime power. For any positive integers n and m , there exists a Δ -universal $(N; q^n, q^m)$ -hash family, where $N = \max\{q^n, q^m\}$.

Using ϵ - Δ -universal hash families, it is very easy to construct partitioned ϵ -universal ones as the following lemma shows.

Lemma 2. If there is an ϵ - Δ -universal $(N; u, r)$ -hash family, then there is also a partitioned ϵ -universal $(N; ur, r)$ -hash family.

Proof. Let $\mathcal{H} : U \rightarrow R$ be an ϵ - Δ -universal hash family where $|U| = u$ and $|R| = r$. For each $h \in \mathcal{H}$ we define the mapping $f_h : U \times R \rightarrow R$, $(x_1, x_2) \mapsto h(x_1) + x_2$. Then $\mathcal{F} = \{f_h \mid h \in \mathcal{H}\}$ is the desired partitioned ϵ -universal $(N; ur, r)$ -hash family. To see this, define the equivalence relation \sim on $U \times R$ as $(x_1, x_2) \sim (x'_1, x'_2) \Leftrightarrow x_1 = x'_1$. Clearly, two different keys (x_1, x_2) and (x'_1, x'_2) collide if and only if $h(x_1) - h(x'_1) = x'_2 - x_2$. Because \mathcal{H} is ϵ - Δ -universal, this is for $x_1 \neq x'_1$ the case with a probability of at most ϵ . If $x_1 = x'_1$, then both keys belong to the same equivalence class and they differ only in the second component. Thus, they do not collide under any function in \mathcal{F} . Since the size of each equivalence class is r , \mathcal{F} is partitioned ϵ -universal. \square

It is well-known that u/r is a lower bound on the size N of a $(1/r)$ -universal $(N; u, r)$ -hash family (see e.g. [16]). This is clearly not a tight lower bound for the case $r < u < r^2$, since for $r < u$, there is at least one key pair whose collision probability is not 0, and thus at least r hash functions are required to obtain a collision probability of at most $1/r$.

Definition 7. A (partitioned) universal $(N; u, r)$ -hash family is called minimal if $N = 1$ in the case $u \leq r$ and $N = \max\{u/r, r\}$ in the case $u > r$.

Note that for $u \leq r$, a minimal partitioned universal $(N; u, r)$ -hash family is the set consisting of the identity, only. Using Lemma 1 and Lemma 2 for the case $u > r$, we obtain minimal partitioned universal hash families for all u and r which are powers of the same prime.

Corollary 3. *Let q be a prime power. For any positive integers n and m , there exists a minimal partitioned universal $(N; q^n, q^m)$ -hash family.*

We present now two construction methods, which can later be combined to recursively construct the minimal OU hash families from Theorem 1. More precisely, in the following two lemmas we show that we can combine a partitioned universal $(N_1; u, r)$ -hash family either with an OU $(N_2; r, r^2/u)$ -hash family (in the case $r < u \leq r^2$) or with an OU $(N'_2; u/r, r)$ -hash family (in the case $u \geq r^2$) in order to obtain an OU $(N; u, r)$ -hash family.

Lemma 3. *Let $u = k^2 \cdot \ell$ and $r = k \cdot \ell$. If \mathcal{G} is a partitioned universal $(N_1; k^2\ell, k\ell)$ -hash family and \mathcal{F} is an optimally universal $(N_2; k\ell, \ell)$ -hash family, then there is an optimally universal $(N; u, r)$ -hash family \mathcal{H} , where $N = N_1N_2(u-1)/\gcd(N_1(r-1), N_2(u-r))$. Moreover, if \mathcal{G} and \mathcal{F} are both minimal, then so is \mathcal{H} .*

Proof. Let K and L be sets of cardinality k and ℓ , resp. We may assume w.l.o.g. that \mathcal{G} consists of mappings $K \times R \rightarrow R$, where $R = K \times L$, and \mathcal{F} consists of mappings $K \times L \rightarrow L$. Let $z = \text{lcm}(N_1(r-1), N_2(u-r))$ and $z_1 = z/(N_1(r-1))$ as well as $z_2 = z/(N_2(u-r))$. Let the family \mathcal{G}' consist of z_1 copies of each hash function in \mathcal{G} and the family \mathcal{F}' consist of z_2 copies of the hash functions in \mathcal{F} . For each function $f \in \mathcal{G}' \cup \mathcal{F}'$ we define the mapping

$$h_f : K \times R \rightarrow K \times L, \quad (x_1, x_2) \mapsto \begin{cases} f(x_1, x_2) & \text{if } f \in \mathcal{G}', \\ (x_1, f(x_2)) & \text{if } f \in \mathcal{F}'. \end{cases}$$

Finally, the hash family \mathcal{H} consists of the functions h_f with $f \in \mathcal{G}' \cup \mathcal{F}'$.

A somewhat technical computation (which we omit here due to space restrictions) shows that the cardinality of \mathcal{H} is in fact N and that \mathcal{G} and \mathcal{F} being minimal implies that $N = (u-1)/\gcd(u-1, r-1)$ (i.e., \mathcal{H} has the size of a minimal OU hash class). Hence, it remains to show that \mathcal{H} is optimally universal. Let $c = |\mathcal{G}'|/|\mathcal{F}' \cup \mathcal{G}'|$, which is the probability, that a randomly chosen $f \in \mathcal{G}' \cup \mathcal{F}'$ is an element of \mathcal{G}' . Hence,

$$c = \frac{z_1 N_1}{z_2 N_2 + z_1 N_1} = \frac{1/(r-1)}{1/(u-r) + 1/(r-1)} = \frac{u-r}{r-1+u-r} = \frac{u-r}{u-1}. \quad (1)$$

Thus, by definition of optimally universality it suffices to show that any two distinct keys $x = (x_1, x_2)$ and $x' = (x'_1, x'_2)$ in $K \times R$ have a collision probability of at most c/r .

Assume first $x_1 = x'_1$, thus $x_2 \neq x'_2$. Since in this case x and x' are elements of the same equivalence class with respect to the partitioned universality of \mathcal{G}' ,

they do not collide under any function h_f with $f \in \mathcal{G}'$. Under the condition that $f \in \mathcal{F}'$, it follows from \mathcal{F}' being OU that $h_f(x)$ equals $h_f(x')$ with a probability of at most $(r - \ell)/(\ell(r - 1))$. Therefore, we have for randomly chosen $f \in \mathcal{G}' \cup \mathcal{F}'$ a total collision probability of at most

$$(1 - c) \cdot \frac{r - \ell}{\ell(r - 1)} = \frac{r - 1}{u - 1} \cdot \frac{r - \ell}{\ell(r - 1)} = \frac{r - \ell}{\ell(u - 1)} = \frac{rk - \ell k}{k\ell(u - 1)} = \frac{u - r}{r(u - 1)} = \frac{c}{r}.$$

Let now $x_1 \neq x'_1$. Under the condition that f was chosen from \mathcal{G}' , the collision probability of x and x' is at most $1/r$. If on the other hand f was chosen from \mathcal{F}' , then the keys do not collide at all, which follows straight from the definition of h_f . Therefore, we again have a total collision probability (for f chosen randomly from $\mathcal{G}' \cup \mathcal{F}'$) of at most c/r . \square

Lemma 4. *Let $u = kr$ and $k \geq r$. If \mathcal{G} is a partitioned universal $(N_1; u, r)$ -hash family and \mathcal{F} is an optimally universal $(N_2; k, r)$ -hash family, then there exists an optimally universal $(N; u, r)$ -hash family \mathcal{H} , where $N = N_1 N_2 (ur - r) / \gcd(N_1(u - r), N_2(ur - u))$. Moreover, if \mathcal{G} and \mathcal{F} are both minimal, then so is \mathcal{H} .*

Proof. Let K be a set of size k and assume w.l.o.g. that \mathcal{G} and \mathcal{F} consist of mappings $K \times R \rightarrow R$ and $K \rightarrow R$, resp. Let $z = \text{lcm}(N_1(u - r), N_2(ur - u))$ and $z_1 = z/(N_1(u - r))$ as well as $z_2 = z/(N_2(ur - u))$. Further, let the family \mathcal{G}' consist of z_1 copies of each function in \mathcal{G} and the family \mathcal{F}' consist of z_2 copies of each function in \mathcal{F} . For each $f \in \mathcal{G}' \cup \mathcal{F}'$ we define the hash function

$$h_f : K \times R \rightarrow R, \quad (x_1, x_2) \mapsto \begin{cases} f(x_1, x_2) & \text{if } f \in \mathcal{G}', \\ f(x_1) & \text{if } f \in \mathcal{F}'. \end{cases}$$

Finally, let \mathcal{H} be the family of functions h_f with $f \in \mathcal{G}' \cup \mathcal{F}'$.

A straightforward computation (which we omit here due to space restrictions) shows that $|\mathcal{H}| = N$ and that $N = (u - 1)/\gcd(u - 1, r - 1)$ if \mathcal{G} and \mathcal{F} are minimal. It remains to show that \mathcal{H} is optimally universal. Let $\epsilon = |\mathcal{F}'|/|\mathcal{F}' \cup \mathcal{G}'|$. Then

$$\epsilon = \frac{z_2 N_2}{z_1 N_1 + z_2 N_2} = \frac{1/(ur - u)}{1/(u - r) + 1/(ur - u)} = \frac{u - r}{ur - u + u - r} = \frac{u - r}{ur - r}.$$

Thus, it suffices to show that any two distinct keys $x = (x_1, x_2)$ and $x' = (x'_1, x'_2)$ in $K \times R$ collide under a randomly chosen function in \mathcal{H} with a probability of at most ϵ . Assume first that $x_1 = x'_1$ and thus $x_2 \neq x'_2$. Then, by partitioned universality of \mathcal{G} , x and x' do not collide under any function in \mathcal{G}' , but under all functions in \mathcal{F}' . Therefore, the collision probability is exactly ϵ . Let now $x_1 \neq x'_1$. Under the condition $f \in \mathcal{G}'$ the keys collide with a probability of at most $1/r$. If on the other hand $f \in \mathcal{F}'$, then the collision probability is at most $(k - r)/(r(k - 1))$ since \mathcal{F}' is optimally universal. Therefore, the probability that $h_f(x)$ equals $h_f(x')$ for a randomly chosen $f \in \mathcal{H}$ is bounded above by

$$(1 - \epsilon) \cdot \frac{1}{r} + \epsilon \cdot \frac{k - r}{kr - r} = \left(1 - \frac{u - r}{ur - r}\right) \cdot \frac{1}{r} + \frac{u - r}{ur - r} \cdot \frac{u/r - r}{u - r} = \frac{u - r}{ur - r} = \epsilon. \quad \square$$

We can now combine the two construction methods in order to prove our main result.

Proof of Theorem 1: Let q be an arbitrary prime power. We show by induction on n that for all $1 \leq m \leq n$ there exists a minimal optimally universal hash family $U \rightarrow R$, where U and R are sets of cardinalities $u = q^n$ and $r = q^m$, resp. In order to make clear, that this is not merely a proof of the existence of such a hash family, but rather an explicit construction, we describe such a minimal optimally universal $(N; u, r)$ -hash family $\mathcal{H}_{n,m}^q$.

We use as the universe and range the extension fields $U = (\mathbb{F}_q)^n$ and $R = (\mathbb{F}_q)^m$, resp. Let for all $m \leq n$ the hash family $\mathcal{G}_{n,m}^q$ consist of the functions $f_a : (\mathbb{F}_q)^{n-m} \times (\mathbb{F}_q)^m \rightarrow (\mathbb{F}_q)^m$, $(x_1, x_2) \mapsto \phi(a \cdot x_1) + x_2$, where $\phi : (\mathbb{F}_q)^{n-m} \rightarrow (\mathbb{F}_q)^m$ is a projection from $(\mathbb{F}_q)^{n-m}$ to m arbitrary coordinates of the extension field. Note that for the computation of f_a the multiplication is in the field $(\mathbb{F}_q)^{n-m}$ while the addition is in $(\mathbb{F}_q)^m$. According to the discussion at the beginning of this section $\mathcal{G}_{n,m}^q$ is minimal partitioned universal.

If $m = n$, then $\mathcal{H}_{n,m}^q$ contains only the identity $id : (\mathbb{F}_q)^n \rightarrow (\mathbb{F}_q)^n$, which is obviously minimal optimally universal. Hence, for $n = 1$ we obtain a trivial hash family $\mathcal{H}_{1,1}^q$. Let now $1 \leq m < n$ and assume that the hash families $\mathcal{H}_{m',n'}^q$ have been constructed for all $1 \leq m' \leq n' < n$. In the case $m < n \leq 2m$, we choose $\mathcal{H}_{n,m}^q$ as the union of z_1 copies of $\mathcal{G}_{n,m}^q$ and z_2 copies of all mappings $(\mathbb{F}_q)^{n-m} \times (\mathbb{F}_q)^m \rightarrow (\mathbb{F}_q)^{n-m} \times (\mathbb{F}_q)^{2m-n}$, $(x_1, x_2) \mapsto (x_1, f(x_2))$, with $f \in \mathcal{H}_{m,2m-n}^q$, where z_1 and z_2 are the integers determined in the proof of Lemma 3. In the case $n > 2m$, we choose $\mathcal{H}_{n,m}^q$ as the union of z'_1 copies of $\mathcal{G}_{n,m}^q$ and z'_2 copies of the mappings $(\mathbb{F}_q)^{n-m} \times (\mathbb{F}_q)^m \rightarrow (\mathbb{F}_q)^m$, $(x_1, x_2) \mapsto f(x_1)$, with $f \in \mathcal{H}_{n-m,m}^q$, where z'_1 and z'_2 are the integers determined in the proof of Lemma 4. Then, according to the proofs of Lemma 3 and Lemma 4 and by the induction hypothesis, $\mathcal{H}_{n,m}^q$ is minimal optimally universal. \square

We finally remark that the hash functions from $\mathcal{H}_{n,m}^q$ may in fact be evaluated by simple finite field arithmetic. By the above proof it is easy to see that each hash function $h \in \mathcal{H}_{n,m}^q$ has the form

$$g_{i,j}(x_1, \dots, x_n) \mapsto (x_1, \dots, x_i, f(x_{i+1}, \dots, x_j)),$$

where either $i = j$ and $g_{i,j}$ is the identity, or $i < j$ and f is a function in $\mathcal{G}_{j-i,m-i}^q$. Hence, h can essentially be evaluated by one finite field multiplication and one finite field addition. Although the evaluation of these hash functions is simple, there seems to be no obvious uniform algorithm allowing us to choose a function from $\mathcal{H}_{n,m}^q$ in constant time. Nevertheless, it is easy to enumerate all hash functions in $\mathcal{H}_{n,m}^q$ in time $|\mathcal{H}_{n,m}^q|$. This means also that the blocks of the corresponding RBIBD $_{\lambda}[k; v]$ can be constructed in linear time (with respect to the size of their description) ordered by the parallel classes (recall that a parallel class in the RBIBD corresponds to a hash function in the hash family). Therefore, the algorithm of Hofmeister and Lefmann can be used to compute a k -cut in a graph as stated in Corollary 2 and our modified algorithm has in fact the running time stated in Theorem 3.

Acknowledgements

The author thanks Malcolm Greig for answering questions regarding RBIBDs and Ingo Wegener for helpful comments on this text.

References

1. M. Atici and D. R. Stinson. Universal hashing and multiple authentication. In *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pp. 16–30. 1996.
2. T. Beth, D. Jungnickel, and H. Lenz. *Design Theory*, volume 1. Cambridge University Press, second edition, 1999.
3. J. Bierbrauer. Universal hashing and geometric codes. *Designs, Codes and Cryptography*, 11:207–221, 1997.
4. J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979.
5. C. J. Colbourn and J. H. Dinitz, editors. *The CRC Handbook of Combinatorial Designs*. CRC Press, first edition, 1996.
6. M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25:19–51, 1997.
7. M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM Journal on Computing*, 23:738–761, 1994.
8. M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the Association for Computing Machinery*, 31:538–544, 1984.
9. T. Hofmeister and H. Lefmann. A combinatorial design approach to MAXCUT. *Random Structures and Algorithms*, 9:163–173, 1996.
10. M. Holland and G. Gibson. Parity declustering for continuous operation in redundant disk arrays. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 23–35. 1992.
11. Y. Mansour, N. Nisan, and P. Tiwari. The computational complexity of universal hashing. *Theoretical Computer Science*, 107:121–133, 1993.
12. V. Nguyen and Z. Tuza. Linear-time approximation algorithms for the max cut problem. *Combinatorics, Probability and Computing*, 2:201–210, 1993.
13. D. V. Sarwate. A note on universal classes of hash functions. *Information Processing Letters*, 10:41–45, 1980.
14. E. J. Schwabe and I. M. Sutherland. Flexible usage of redundancy in disk arrays. *Theory of Computing Systems*, 32:561–587, 1999.
15. D. R. Stinson. Combinatorial techniques for universal hashing. *Journal of Computer and System Sciences*, 48:337–346, 1994.
16. D. R. Stinson. Universal hashing and authentication codes. *Designs, Codes and Cryptography*, 4:369–380, 1994.
17. D. R. Stinson. On the connections between universal hashing, combinatorial designs and error-correcting codes. *Congressus Numerantium*, 114:7–27, 1996.

Towards Constructing Optimal Strip Move Sequences

Meena Mahajan¹, Raghavan Rama², and S. Vijayakumar^{1,*}

¹ The Institute of Mathematical Sciences, Chennai 600 113, India

`{meena,vjy}@imsc.res.in`

² Department of Mathematics, Indian Institute of Technology, Madras,

Chennai 600 036, India

`ramar@iitm.ac.in`

Abstract. The Sorting by Strip Moves problem, SBSM, was introduced in [6] as a variant of the well-known Sorting by Transpositions problem. A restriction called Block Sorting was shown in [2] to be NP-hard. In this article, we improve upon the ideas used in [6] to obtain a combinatorial characterization of the optimal solutions of SBSM. Using this, we show that a strip move which results in a permutation of two or three fewer strips or which exchanges a pair of adjacent strips to merge them into a single strip necessarily reduces the strip move distance. We also establish that the strip move diameter for permutations of size n is $n - 1$. Further, we exhibit an optimum-preserving equivalence between SBSM and the Common Substring Removals problem (CSR) – a natural combinatorial puzzle. As a consequence, we show that sorting a permutation via strip moves is as hard (or as easy) as sorting its inverse.

1 Introduction

The Sorting by Strip Moves problem, SBSM, was introduced in [6] as a variant of the well-known Sorting by Transpositions problem SBT [1, 4, 5, 7]. Let π be a string which is a permutation of a finite set, say $\{1, 2, \dots, n\}$. A transposition is the operation of picking up a substring of π and placing it elsewhere in the string. The SBT problem asks for a shortest sequence of transpositions that can transform π into the identity permutation $\text{id}_n = 1\ 2 \dots n$. A strip move is a restriction of a transposition: the substring being moved must be a maximal substring of π which is also a substring of id_n . Such maximal substrings are called strips; for example, in the permutation $5\ 2\ 3\ 4\ 1\ 6$ on 6 elements, there are four strips $[5]$, $[234]$, $[1]$ and $[6]$. The Sorting by Strip Moves problem, SBSM asks for a shortest sequence of strip moves that can transform π to id_n .

Analyzing and understanding strip moves combinatorially is worthwhile for two reasons. Firstly, strips are an important combinatorial object in understanding transpositions. In particular, strip move sorting directly gives a factor 3

* Part of this work was done when this author was with the Department of Mathematics, IIT Madras, India.

approximation for sorting by transpositions [6], and there are optimal transposition sequences which do not, at any stage, break existing strips [3]. Secondly, a restriction, sorting by canonical strip moves (move a strip to a place where it merges with another strip) is shown in [2] to be NP-complete, while [6] shows that it is actually equivalent to SBSM. While [2] mentions various heuristic approaches to solving SBSM, and provides sufficient conditions under which a sorting sequence is optimal, [6] devises a 2-approximation algorithm for SBSM using certain combinatorial properties about optimal strip move sequences.

In this paper, we continue the task of exploring the combinatorial structure of optimal strip move sequences. We show that the strip move distance $s(\pi)$ of a permutation π equals the size of a largest *compatible edge set* in an associated *order graph* G_π (Theorem 1). This allows us to establish the strip-move diameter (Corollary 2) for permutations of size n ; note that the corresponding question for general transpositions is still unsettled. Using our characterization, we establish that certain strip moves are optimal: they necessarily reduce the strip move distance. In particular, a strip move that reduces the number of strips in the permutation by 2 or 3 is optimal (Theorem 2). Also, if two adjacent strips are consecutive but out of order, then the strip move that exchanges them is optimal (Theorem 3). Thus, for instance, $s(\boxed{7\ 8}\boxed{4}\boxed{6}\boxed{5}\boxed{1\ 2\ 3}\boxed{9}) > s(\boxed{7\ 8}\boxed{4\ 5\ 6}\boxed{1\ 2\ 3}\boxed{9}) > s(\boxed{7\ 8}\boxed{1\ 2\ 3\ 4\ 5\ 6}\boxed{9})$.

We also establish the equivalence of Sorting By Strip Moves and the following combinatorial puzzle. Let π, ϕ be a pair of permutations of the same length, viewed as strings, with $\pi \neq \phi$. We wish to quantify how different these strings are. One way to do this is the following: identify a substring common to both π and ϕ and delete it. Repeat this process until both strings are the same. How many such deletions are needed? We denote the minimum number of deletions needed by $csr(\pi, \phi)$, and we define the *Common Substring Removals* problem, CSR, as the problem of computing $csr(\pi, \phi)$ for a given π, ϕ . For example, consider the pair $\pi = 3\ 5\ 1\ 4\ 2$ and $\phi = 1\ 4\ 3\ 2\ 5$ from S_5 . An optimal common substring removal sequence that makes this pair identical is $(\mathbf{3}\ 5\ 1\ 4\ 2, 1\ 4\ \mathbf{3}\ 2\ 5) \rightarrow (5\ \mathbf{1}\ 4\ \mathbf{2}, 1\ 4\ \mathbf{2}\ 5) \rightarrow (5, 5)$. We show that SBSM and CSR are computationally equivalent (Theorem 4). The reductions preserve optimal values; thus the 2-approximation algorithm of [6] for SBSM is also a 2-approximation algorithm for the CSR problem. An interesting consequence of this equivalence is that sorting a permutation via strip moves is as hard (or as easy) as sorting its inverse (Corollary 3). The analogous result for transpositions is quite obvious, but, given the way strip moves are defined, there is no direct way of seeing this for strip move sorting.

Some of our results are subsumed by results of [2, 6], but the proof technique used here is completely different.

Due to space constraints, some proofs are omitted. See [8] for details.

2 Some Preliminaries

We follow notation from [6]. We denote by $[n]$ the set $\{1, 2, \dots, n\}$, by S_n the set of all permutations over $[n]$, and by id_n the identity permutation on n elements.

For $\pi \in S_n$, a *strip* of π is a maximal common substring of π and id_n . A *substrip* is a substring of a strip. If a strip α is the substring $u \ u + 1 \ \dots \ u + t$ for some u, t , then the strip containing $u - 1$ (or $u + t + 1$) is said to be the *predecessor* (*successor*, respectively) of α . Moving α to the immediate right of $u - 1$ or to the immediate left of $u + t + 1$ is called a *canonical strip move*. (If $u = 1$ or $u + t = n$, then moving α to the extreme left or right respectively is also a canonical strip move.) The *strip move distance* $s(\pi)$ is the minimum number of strip moves needed to sort π ; the corresponding minimum if strip moves are required to be canonical is the *canonical strip move distance* $sc(\pi)$.

Given a permutation π , replace each strip x by any one representative element to obtain a sequence S . Now, replacing each $a \in S$ by its rank in S , we obtain a permutation on k elements, where k is the number of strips in π . This permutation is called the *kernel* of π , denoted $\ker(\pi)$.

It is known that $s(\pi) = sc(\pi) = s(\ker(\pi))$ [6] and that computing $sc(\pi)$ is NP-hard [2].

3 Compatible Edge Sets and Optimal Solutions of SBSM

Given a permutation $\pi \in S_n$, we define an order graph G_π associated with it. We then define a property, which we call compatibility, of subsets of edges of G_π . We establish that optimally sorting a permutation π by strip moves is equivalent to finding a largest compatible edge set in G_π .

Definition 1. *The order graph of a permutation $\pi \in S_n$ is the directed graph $G_\pi = (V_\pi, E_\pi)$ where $V_\pi = [n]$ and $E_\pi = \{(u, v) \mid (u < v) \wedge (\pi_u^{-1} < \pi_v^{-1})\} = \{(\pi_i, \pi_j) \mid (i < j) \wedge (\pi_i < \pi_j)\}$.*

Definition 2. 1. *Two edges (u, v) and (w, x) of E_π are said to interleave by value if $u \leq w < v \leq x$. They are said to interleave by position if $\pi_u^{-1} \leq \pi_w^{-1} < \pi_v^{-1} \leq \pi_x^{-1}$. They are said to interleave if they interleave either by value or by position (or by both).*

2. *An edge (u, v) is said to contain an edge (w, x) of E_π if $u < w < x < v$ (contain by value) or $\pi_u^{-1} < \pi_w^{-1} < \pi_x^{-1} < \pi_v^{-1}$ (contain by position).*

3. *For any $C \subseteq E_\pi$, the inclusion graph $G(C, \pi)$ corresponding to C is the directed graph (C, A) where $A = \{((u, v), (w, x)) \mid (u, v) \text{ contains } (w, x)\}$.*

4. *A set $C \subseteq E_\pi$ is said to be compatible if no two edges of C interleave and if $G(C, \pi)$ is acyclic.*

5. *c_π is the size of any compatible set of edges of maximum cardinality.*

This definition of compatible edge sets generalises that of *non-crossing sets*, introduced in [6] to efficiently solve a certain variant of SBSM exactly.

Note that if C is a compatible edge set, then the subgraph $([n], C)$ is a collection of vertex-disjoint directed paths. Also, if C is a compatible edge set and $C' \subseteq C$, then C' is also compatible.

The main result of this section is that for all $\pi \in S_n$, $n \geq 1$, $s(\pi) = n - 1 - c_\pi$. We need to establish several lemmas to finally prove this theorem.

We call (u, v) a *unit edge* if $v - u = \pi_v^{-1} - \pi_u^{-1} = 1$. Thus unit edges are exactly those edges of the order graph G_π which link the adjacent elements of a strip. The following is easy to see.

Lemma 1. *Every compatible set in E_π of size c_π contains all unit edges of G_π .*

Definition 3. *For a permutation π , let C be any compatible subset of E_π . A strip α of π is said to be free with respect to C if every edge of C has neither or both endpoints in α .*

Lemma 2. *For a permutation π , let C be a compatible subset of E_π of size c_π , and let α be a strip of π free with respect to C . Moving α gives a permutation σ with $c_\sigma \geq c_\pi$. Furthermore, if α is moved canonically (to predecessor or successor) to obtain σ , then $c_\sigma > c_\pi$.*

Proof. Since α is a free strip, C is a subset of E_σ as well. Clearly, it has no edges interleaving by value. No edges interleave by new position either, since even in the new position no edge has just one endpoint in α . And the graph $G(C, \sigma)$ is acyclic as well, since the edges other than those in α form no cycles, and the unit edges within α do not contain any other edge by value or by new position. So C is a compatible subset of E_σ . Hence $c_\sigma \geq |C| = c_\pi$.

Further, if α is moved canonically, then G_σ has an extra unit edge created by this move, which is not even in C . So by Lemma 1, $|C| < c_\sigma$. Thus $c_\sigma > c_\pi$. \square

Lemma 3. *If σ is obtained from π via a strip/substrip move, then $c_\sigma \geq c_\pi - 1$. Furthermore, if σ is obtained from π via a canonical strip move, then $c_\sigma \geq c_\pi$.*

Proof. Let C be a compatible set of edges in G_π of size c_π . By Lemma 1, it contains all the unit edges of G_π . We now show that there is a compatible edge set C' of σ with $|C'| \geq |C| - 1$.

Let α be the strip or substrip moved in order to obtain σ from π . Let m be the number of edges of C with one endpoint in α and one endpoint outside α . Clearly, $m \in \{0, 1, 2\}$. If $m = 0$, then α is a free strip with respect to C , and the result follows from Lemma 2. So now assume that $m > 0$.

Case 1: $m=1$. Let (u, v) be the single edge of C with an endpoint in α and an endpoint outside it. (If α is a proper substrip, then this is in fact a unit edge.) As argued in the proof of Lemma 2, $C' = C \setminus \{(u, v)\}$ is a compatible edge set in E_π and E_σ , showing that $c_\sigma \geq c_\pi - 1$.

Further, if α is moved canonically, then G_σ has an extra unit edge created by this move, which is not in C' . So by Lemma 1, $|C'| < c_\sigma$ and hence $c_\sigma \geq c_\pi$.

Case 2: $m=2$. Let α be the (sub)strip $v+1, v+2, \dots, v+k$, with $\pi_{v+j}^{-1} = i+j$ for $j \in [k]$. Since C contains all the unit edges of G_π , the two edges straddling α must be of the form $(u, v+1)$ and $(v+k, w)$. Let $C' = C \setminus \{(u, v+1), (v+k, w)\}$, and $C'' = C' \cup \{(u, w)\}$. As argued above, C' is a compatible edge set in E_π and in E_σ . (In particular, $G(C', \sigma)$ is acyclic.) We now show that C'' is a compatible edge set of G_σ , proving that $c_\sigma \geq c_\pi - 1$. We only need to show that adding (u, w) keeps this set compatible.

C has a path $\rho = u, v+1, v+2, \dots, v+k, w$ which is replaced in C'' by two paths: a single-edge path $\rho' = u, w$ and the path $\rho'' = v+1, v+2, \dots, v+k$; all other edges of C and C'' are the same. Since no other edge of C can interleave with ρ by value, it is clear that no other edge of C'' can interleave with paths ρ' and ρ'' by value. Since the relative positions of all elements other than $v+1, \dots, v+k$ is unchanged, no edge of C and C'' can interleave with ρ' by position either. So C'' has no interleaving edges.

We must now show that $G(C'', \sigma)$ is acyclic. Assume it is not; then any cycle θ in it must pass through the vertex (u, w) , since we already know that $G(C', \sigma)$ is acyclic. Since the moved unit edges corresponding to α do not contain any edge by value or by position, none of the corresponding vertices can figure in θ . Let (x_1, y_1) be predecessor and (x_2, y_2) the successor of (u, w) in θ . Then (x_1, y_1) must contain all edges of ρ in E_π . And either $(u, v+1)$ or $(v+k, w)$, both edges of ρ , must contain (x_2, y_2) in E_π . The remaining edges of θ are present in $G(C, \pi)$ as well. Thus, corresponding to θ , there is a cycle θ' in $G(C, \pi)$ as well, contradicting the compatibility of C .

Further, if α is moved canonically, then G_σ has an extra unit edge created by this move, which is not in C'' . So by Lemma 1, $|C''| < c_\sigma$ and hence $c_\sigma \geq c_\pi$. \square

Lemma 4. *If π is obtained from σ by a strip move, then $c_\pi \leq c_\sigma + 1$.*

Proof. Given σ , let π be obtained through a move of some strip α on σ . Then α is either a strip or a substrip in π , and so, moving it back to its original place is a strip or substrip move from π yielding σ . Now apply Lemma 3. \square

Lemma 5. *Given $\pi \neq \text{id}_n$ and a compatible set $C \subseteq E_\pi$ of size c_π , we can find a canonical strip move ρ on π leading to a σ with $c_\sigma = c_\pi + 1$.*

Proof. By Lemmas 2 and 4, it suffices to show that there exists a strip α free with respect to C . Consider the inclusion graph $G(C, \pi)$; it is acyclic by definition. All unit edges of G_π have zero out-degree in $G(C, \pi)$. Let C' be the set of non-unit edges of C ; C' is also a compatible set. If $C' = \emptyset$, then C consists only of unit edges of G_π . Therefore, all strips of π are free with respect to C . Otherwise, consider the subgraph G' of $G(C, \pi)$ induced by the vertex set C' . By heredity, G' is also acyclic. Choose any vertex $(u, v) \in C'$ of zero out-degree in G' ; let $i = \pi_u^{-1}$, $j = \pi_v^{-1}$. Let A be the set $\{u+1, \dots, v-1\} \cup \{\pi_{i+1}, \dots, \pi_{j-1}\}$. Since (u, v) is not a unit edge, A is non-empty. Since (u, v) is of out-degree zero in G' , it contains no non-unit edge of C . So the strips containing any element of A must all be free with respect to C . \square

We now prove the main theorem of this section.

Theorem 1. *For all $\pi \in S_n$, $n \geq 1$, $s(\pi) = n - 1 - c_\pi = sc(\pi)$.*

Proof. We will show that (i) $s(\pi) \geq n - 1 - c_\pi$, and (ii) $sc(\pi) \leq n - 1 - c_\pi$. Since $s(\pi) \leq sc(\pi)$, the result follows.

To prove (i), let $\rho_1, \rho_2, \dots, \rho_k$ be a sequence of strip moves sorting π . Let $\pi^0 = \pi$, and π^i be the permutation obtained by applying strip move ρ_i to

π^{i-1} . Let c_i denote c_{π^i} . By Lemma 4, $c_i \leq c_{i-1} + 1$. Since $\pi^k = \text{id}_n$, and since $c_{\text{id}_n} = n - 1$, we have $n - 1 = c_k \leq c_{k-1} + 1 \leq \dots \leq c_1 + (k - 1) \leq c_0 + k = c_\pi + k$, hence $k \geq n - 1 - c_\pi$. This holds for any strip move sorting sequence; in particular, it holds for an optimal sequence with $k = s(\pi)$. Hence $s(\pi) \geq n - 1 - c_\pi$.

To prove (ii), repeatedly apply Lemma 5 and note that $c_{\text{id}_n} = n - 1$. \square

Corollary 1. $s(\pi) = s(\ker(\pi))$.

If $\pi \in S_n$ and $\pi \neq \text{rev}_n = n \ n - 1 \dots 2 \ 1$, then clearly $c_\pi \geq 1$, and hence $s(\pi) \leq n - 2$. For $\pi = \text{rev}_n$, $c_\pi = 0$, and so $s(\text{rev}_n) = n - 1$. Thus we have:

Corollary 2. $D(n) = \max\{s(\pi) | \pi \in S_n\} = n - 1$.

Lemma 3 and Theorem 1 imply that canonical strip moves do not increase the strip-move distance. In the next two sections, we identify some canonical strip moves which provably reduce it.

4 2-Moves and 3-Moves Are Provably Optimal

In this section, we establish that strip moves which reduce the number of strips by 2 or more necessarily reduce the strip move distance. This result is proved in many stages.

Lemma 6. *If $\pi \in S_n$ has $\pi_i = u - 1$ and $\pi_{i+1} = v$ for some i and some $u < v$, and if the elements $u, u + 1, u + 2, \dots, v - 1$ form a strip α of π , then the permutation σ obtained from π by moving α to between u and v has $s(\sigma) < s(\pi)$.*

Proof. From Corollary 1, $s(\pi) = s(\ker(\pi))$. In $\ker(\pi)$, α is a single element. So it suffices to prove the lemma when $|\alpha| = 1$, i.e. $v = u + 1$ and $\alpha = u$.

Let π have $\pi_i = u - 1$ and $\pi_{i+1} = u + 1$. By Lemma 2 and Theorem 1, it suffices to prove that there is some compatible edge set of size c_π with respect to which the strip containing u is free.

Let C be any compatible subset of E_π of size c_π . If u is free with respect to C there is nothing to prove. Otherwise, without loss of generality, assume that $k = \pi_u^{-1} > i + 1$; the situation where $k < i$ is symmetric. There are three cases.

Case 1: There is a single edge (u, w) in C incident on u . Then C has no edge of the form $(x, u + 1)$, since such an edge interleaves by value with (u, w) . If C does not have any edge $(u - 1, x)$, then let $C' = C \setminus \{(u, w)\} \cup \{(u - 1, u + 1)\}$. Otherwise let $(u - 1, x)$ be an edge in C and let y be the rightmost endpoint of the path in C containing $u + 1$. Let $C' = C \setminus \{(u, w), (u - 1, x)\} \cup \{(u - 1, u + 1), (y, x)\}$. It can be seen that C' is a compatible subset of E_π of size c_π . And u is free with respect to C' .

Case 2: There is a single edge (w, u) in C incident on u . Then $(u - 1, u + 1)$ cannot be in C . If C has no edge with right endpoint $u + 1$, then the set $C' = C \setminus \{(w, u)\} \cup \{(u - 1, u + 1)\}$ is a compatible subset of E_π of size c_π , satisfying Case 1 above. Otherwise, let $(x, u + 1)$ be an edge in C . To avoid interleaving by value with (w, u) , we have $x < w$. So $(x, u + 1)$ contains (w, u) by value.

If $\pi_w^{-1} < \pi_x^{-1}$, then (w, u) contains $(x, u + 1)$ by position, creating a cycle in $G(C, \pi)$. So $\pi_w^{-1} > \pi_x^{-1}$. But if $\pi_w^{-1} < i$, then $(x, u + 1)$ and (w, u) would interleave by position. So the relative ordering of these elements must be $x, u - 1, u + 1, w, u$. Now the set $C' = C \setminus \{(w, u), (x, u + 1)\} \cup \{(x, u - 1), (u - 1, u + 1)\}$ is a compatible subset of E_π of size c_π , and u is free with respect to C' .

Case 3: There are two edges touching u , (w, u) and (u, x) . There can be no edge into $u + 1$ in C since any such edge will interleave with (u, x) by value. There can be no edge out of $u - 1$ in C since any such edge will interleave with (w, u) by value, unless $w = u - 1$. If $w = u - 1$, set $C' = C \setminus \{(u - 1, u), (u, x)\} \cup \{(u - 1, u + 1), (y, x)\}$ where y is the rightmost endpoint of the path in C containing $u + 1$. Otherwise set $C' = C \setminus \{(w, u), (u, x)\} \cup \{(u - 1, u + 1), (w, x)\}$. Either way, C' is a compatible subset of E_π of size c_π , and u is free with respect to C' . \square

Lemma 7. *If a permutation $\pi \in S_n$ has $\pi_{i-1} = u - 1$ and $\pi_j = u$ for some $u, i, j \geq i + 1, u$, and if the elements $\pi_i, \pi_{i+1}, \pi_{i+2}, \dots, \pi_{j-1}$ form a strip α of π , then the permutation σ obtained from π by moving α canonically has $s(\sigma) < s(\pi)$.*

Proof. Since $s(\pi) = s(\ker(\pi))$, it suffices to prove the lemma when $|\alpha| = 1$.

Let π have $\pi_{i-1} = u - 1$, $\pi_{i+1} = u$, and $\pi_i = v$. By Lemma 2 and Theorem 1, it suffices to prove that there is some compatible edge set of size c_π with respect to which the strip containing v is free.

Let C be a compatible edge set of G_π of size c_π . If v is free with respect to C , there is nothing to prove. If C has a single edge e touching v , then edge $(u - 1, u)$ cannot be in C since it interleaves with e by position. The set $C' = C \setminus \{e\} \cup \{(u - 1, u)\}$ is a compatible set of G_π of size c_π , and v is free with respect to it. If C has two edges (w, v) and (v, x) incident on v , then $C' = C \setminus \{(w, v), (v, x)\} \cup \{(w, x), (u - 1, u)\}$ is a compatible set in G_π of size c_π , with respect to which v is free. \square

Since the goal in SBSM is to move from an unsorted permutation towards id_n which has just one strip, it is natural to prefer strip moves that rapidly reduce the number of strips. With the results we have seen, we can formalise this intuition. Define an x -move to be a strip move that results in a permutation with x fewer strips. (Note that a strip move cannot decrease the number of strips present by more than 3.) Since a 2-move satisfies the assumption of Lemmas 6 or 7 and since a 3-move satisfies the assumption of both, we now have the main result of this section:

Theorem 2. *If a 2-move or a 3-move applied to π gives σ , then $s(\sigma) < s(\pi)$.*

5 Certain 1-Moves Are Provably Optimal

We furnish another application of Theorem 1. We show that a strip move that exchanges a pair of adjacent substrings which are adjacent in the identity permutation can be part of an optimal strip move sorting sequence. Note that such a move may not even be a 2-move. To prove this, we establish an interesting property, which we call *additivity*, of the function $s(\pi)$.

Theorem 3. *If α, β are adjacent strips in π such that $\beta\alpha$ is a substring of id_n , then ϕ obtained from π by exchanging α and β (which is a canonical move of α or β) has $s(\phi) < s(\pi)$.*

Proof. Let the substring $\beta\alpha$ be $u + 1 \ u + 2 \ \dots \ u + t$ for some u, t , and let it occur in positions $i + 1, i + 2, \dots, i + t$ of ϕ for some i . If $\pi_i = u$ or if $\pi_{i+t+1} = u + t + 1$, then exchanging α and β in π is a 2-move or a 3-move, and hence, by Theorem 2, reduces the strip move distance. So now let us assume that $\pi_i \neq u$ and $\pi_{i+t+1} \neq u + t + 1$. Clearly, one way to sort π is to exchange α and β and then sort ϕ ; thus $s(\pi) \leq 1 + s(\phi)$. We show that this is in fact an equality, by defining a notion of *complete* substrings and establishing a more general result concerning them. The proof follows from the statement of Lemma 8 below and the fact that $\alpha\beta$ forms a maximal complete substring in π . \square

Definition 4. *A substring $\pi_{i+1} \dots \pi_{i+t}$ is complete if $\{\pi_{i+1}, \dots, \pi_{i+t}\} = \{u + 1, u + 2, \dots, u + t\}$ for some u . That is, the t consecutive positions $i + 1, i + 2, \dots, i + t$ hold t consecutive integers, though not necessarily in sorted order. If, furthermore, $\pi_i \neq u$ and $\pi_{i+t+1} \neq u + t + 1$, then the substring is said to be maximal complete.*

The strip move distance for the complete substring $\pi_{i+1} \dots \pi_{i+t}$ is defined to be the minimum number of strip moves needed to sort it i.e. to transform it into $u + 1 \dots u + t$.

Lemma 8. *If π contains a maximal complete substring α with $s(\alpha) = q$, then $s(\pi) = q + s(\sigma)$, where σ is the permutation obtained from π by replacing α , in place, with its sorted version.*

Proof. Since α is maximal complete, it is clear that $s(\pi) \leq q + s(\sigma)$. So we only need to show that $s(\pi) \geq q + s(\sigma)$. By Theorem 1, it suffices to show that $c_\sigma \geq q + c_\pi$.

Let $\alpha = \pi_{i+1} \dots \pi_{i+t}$ where $\{\pi_{i+1}, \dots, \pi_{i+t}\} = [u + 1, u + t]$ for some u . Let C be any compatible edge set of G_π of size c_π . Partition C into four subsets as follows:

$$\begin{aligned} C_1 &= \{(v, w) \mid v, w \in [u + 1, u + t]\} && \text{both endpoints in } [u + 1, u + t] \\ C_2 &= \{(v, w) \mid \pi_v^{-1} \leq i, w \in [u + 1, u + t]\} && \text{right endpoint in } [u + 1, u + t] \\ C_3 &= \{(v, w) \mid v \in [u + 1, u + t], \pi_w^{-1} > i + t\} && \text{left endpoint in } [u + 1, u + t] \\ C_4 &= C \setminus (C_1 \cup C_2 \cup C_3) && \text{no endpoint in } [u + 1, u + t] \end{aligned}$$

It follows from Theorem 1 that $|C_1| \leq c_\alpha = t - 1 - q$; i.e. $t - 1 - |C_1| \geq q$. If $|C_2| \leq 1$ and $|C_3| \leq 1$, we construct a compatible set in E_σ of the required size as follows: Let C_5 be the set of $t - 1$ unit edges obtained after sorting α ; $C_5 = \{(u + i, u + i + 1) \mid i \in [t - 1]\}$. Define $C' = C_4 \cup C_5$. Then C' is a subset of E_σ , and it is easy to see that C' is compatible. If $C_2 \cup C_3 = \emptyset$, then $|C_1| + |C_4| = c_\pi$, so C' is the desired set. Otherwise, $C'' = C' \cup \{(v, u + 1) \mid (v, u + i) \in C_2 \text{ for some } i \in [t]\} \cup \{(u + t, w) \mid (u + j, w) \in C_3 \text{ for some } j \in [t]\}$ is the desired set; compatibility and size bounds are easily seen to hold.

If either $|C_2|$ or $|C_3|$ exceeds 1, we show how to construct another compatible subset of E_π of size c_π , in which the corresponding subsets $|C_2|$ and $|C_3|$ are both of size at most 1. Then we can use the above argument.

Assume that $|C_2| = a \geq 2$. Let the edges of C_2 be $(\pi_{i_1}, \pi_{j_1}), (\pi_{i_2}, \pi_{j_2}), \dots, (\pi_{i_a}, \pi_{j_a})$, where $j_1 < j_2 < \dots < j_a$. By compatibility considerations, we have $i_a < \dots < i_2 < i_1 < j_1 < j_2 < \dots < j_a$. Let the unique path in C containing π_{j_h} end at the right at π_{k_h} . From compatibility considerations, it can be argued that $(\pi_{k_h}, \pi_{j_{h+1}}) \in E_\pi$. We define $C'_2 = \{(\pi_{i_1}, \pi_{j_1})\} \cup \{(\pi_{k_h}, \pi_{j_{h+1}}) \mid h \in [a-1]\}$.

If $|C_3| = b \geq 2$, then a set C'_3 of the same size is constructed in a symmetric way. Let the edges of C_3 be $(\pi_{p_1}, \pi_{q_1}), (\pi_{p_2}, \pi_{q_2}), \dots, (\pi_{p_r}, \pi_{q_r})$, where $p_b < \dots < p_2 < p_1$. By compatibility considerations, we have $p_b < \dots < p_2 < p_1 < q_1 < q_2 < \dots < q_b$. Let the unique path in C containing π_{p_h} end at the left at π_{r_h} . Again, we can argue that the edges $(\pi_{p_{h+1}}, \pi_{r_h})$ are indeed present in E_π , and we define $C'_3 = \{(\pi_{p_1}, \pi_{q_1})\} \cup \{(\pi_{p_{h+1}}, \pi_{r_h}) \mid h \in [b-1]\}$;

If both $a, b \geq 2$, then it must be that $j_a < p_b$, since otherwise there will be edges in C interleaving by position.

Clearly, $C'_2 \cup C'_3$ is of the same size as $C_2 \cup C_3$, and is disjoint from $C_1 \cup C_4$. We now show that $C' = C_1 \cup C'_2 \cup C'_3 \cup C_4$ is also a compatible subset of E_π .

It is easy to see that none of the edges of $C'_2 \cup C'_3$ interleave by position. Neither do edges of $C_1 \cup C_4$. And by our choice of C' , an edge of $C'_2 \cup C'_3$ will not interleave by position with an edge of $C_1 \cup C_4$.

If edges of C' interleave by value, then the corresponding edges of C can similarly be shown to interleave by value, since α is a complete substring.

It can be argued that if the inclusion graph $G(C', \pi)$ has a cycle, then so does $G(C, \pi)$, contradicting compatibility of C . \square

Note: The above result can be shown to hold even if α is complete but not maximal.

6 CSR and SBSM Are Equivalent

We first obtain an alternative formulation of SBSM and then show that this alternative formulation is equivalent to CSR. This then implies that sorting a permutation via strip moves is as hard (or as easy) as sorting its inverse.

We say that an increasing substring $a_i \dots a_j$ of a string $S = a_1 a_2 \dots a_m$ of distinct elements from $[n]$ is *tight* if there is no $i \leq k \leq j-1$ such that $a_k < a_l < a_{k+1}$ for some a_l in S . (In other words, if each a_l in S is replaced by its rank $r(a_l)$ in S , then the substring $r(a_i) \dots r(a_j)$ is a substrip in the resulting string.) Let $isr(S)$ denote the length of a shortest sequence of tight increasing substring removals on S that leaves behind an increasing subsequence of S .

For example, consider 3 5 1 4 2. Neither of the increasing substrings $\langle 3, 5 \rangle$ and $\langle 1, 4 \rangle$ is tight, and removing any one element does not make the sequence increasing, so $isr(3\ 5\ 1\ 4\ 2) > 1$. But after deleting 4, the sequence $\langle 3, 5 \rangle$ is tight, and deleting it leaves behind an increasing sequence 1 2. So $isr(3\ 5\ 1\ 4\ 2) = 2$.

The following is easy to see.

Lemma 9. *For any permutation π , $isr(\pi) = s(\pi)$.*

Recall that $csr(\pi, \phi)$ denotes the length of a shortest sequence of common substring removals that makes the pair identical. Given any $\pi, \phi \in S_n$, for any permutation $\psi \in S_n$, $(\psi^{-1}\pi, \psi^{-1}\phi)$ is just a consistent re-labelling for (π, ϕ) . Thus $csr(\pi, \phi) = csr(\psi^{-1}\pi, \psi^{-1}\phi)$. In particular, $csr(\pi, \phi) = csr(\pi^{-1}\phi, id_n) = csr(\phi^{-1}\pi, id_n)$. But if we consider $csr(\sigma, id_n)$, the common substrings removed are always tight increasing substrings. Thus $csr(\sigma, id_n) \geq isr(\sigma)$. In fact, we have equality $csr(\sigma, id_n) = isr(\sigma)$ because the tight increasing substrings removed from σ are necessarily substrings of id_n . These remarks, along with Lemma 9, establish the following theorem.

Theorem 4. *The Sorting by Strip Moves problem SBSM and the Common Substring Removals problem CSR are computationally equivalent.*

For $\pi, \phi \in S_n$, $csr(\pi, \phi) = s(\pi^{-1}\phi) = s(\phi^{-1}\pi)$. For $\sigma \in S_n$, $s(\sigma) = csr(\sigma, id_n)$.

Corollary 3. *For any $\pi \in S_n$, $s(\pi) = s(\pi^{-1})$.*

7 Conclusion

In this article, we have shown that certain types of strip moves are optimal; they necessarily reduce the strip move distance. Analogous results for sorting by transpositions would be very useful. In particular, identifying certain transpositions that provably reduce the transposition distance could guide the way to efficiently sorting by transpositions, either optimally or with good approximation ratios. We believe that the techniques used in this paper can find application to the general sorting by transpositions problem as well.

References

1. V. Bafna and P. Pevzner. Sorting by transpositions. *SIAM Journal on Discrete Mathematics*, 11(2):224–240, 1998.
2. W. W. Bein, L. L. Larmore, S. Latifi, and I. H. Sudborough. Block sorting is hard. *International Journal of Foundations of Computer Science*, 14(3):425–437, 2003.
3. D. A. Christie. *Genome Rearrangement Problems*. PhD thesis, Univ. of Glasgow, 1999.
4. T. Hartman. A simpler 1.5 approximation algorithm for sorting by transpositions. In *Proc. 14th Ann. Sym. Combinatorial Pattern Matching*, Springer LNCS 2676, pp. 156–169, 2003.
5. J. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13:180–210, 1995.
6. M. Mahajan, R. Rama, V. Raman, and S. Vijayakumar. Merging and sorting by strip moves. In *Proc. 23rd Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, Springer LNCS 2914, pp. 314–325, 2003.
7. Pavel Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. MIT Press, Cambridge, MA, USA, 2000.
8. S Vijayakumar. *Sorting with Transpositions*. PhD thesis, Indian Institute of Technology, Madras, April 2004.

Large Triangles in the d -Dimensional Unit-Cube (Extended Abstract)

Hanno Lefmann

Fakultät für Informatik, TU Chemnitz, D-09107 Chemnitz, Germany
lefmann@informatik.tu-chemnitz.de

Abstract. We consider a variant of Heilbronn's triangle problem by asking for fixed dimension $d \geq 2$ for a distribution of n points in the d -dimensional unit-cube $[0, 1]^d$ such that the minimum (2-dimensional) area of a triangle among these n points is maximal. Denoting this maximum value by $\Delta_d^{off-line}(n)$ and $\Delta_d^{on-line}(n)$ for the off-line and the on-line situation, respectively, we will show that $c_1 \cdot (\log n)^{1/(d-1)} / n^{2/(d-1)} \leq \Delta_d^{off-line}(n) \leq C_1 / n^{2/d}$ and $c_2 / n^{2/(d-1)} \leq \Delta_d^{on-line}(n) \leq C_2 / n^{2/d}$ for constants $c_1, c_2, C_1, C_2 > 0$ which depend on d only.

1 Introduction

Given any integer $n \geq 3$, originally Heilbronn's problem asks for the maximum value $\Delta_2(n)$ such that there exists a configuration of n points in the two-dimensional unit-square $[0, 1]^2$ where the minimum area of a triangle formed by three of these n points is equal to $\Delta_2(n)$. For primes n , the points $P_k = 1/n \cdot (k \bmod n, k^2 \bmod n)$, $k = 0, 1, \dots, n-1$, easily show that $\Delta_2(n) = \Omega(1/n^2)$. Komlós, Pintz and Szemerédi [11] improved this to $\Delta_2(n) = \Omega(\log n / n^2)$ and in [5] the authors provide a deterministic polynomial time algorithm achieving this lower bound, which is currently the best known. From the other side, improving earlier results of Roth [14–18] and Schmidt [19], Komlós, Pintz and Szemerédi [10] proved the upper bound $\Delta_2(n) = O(2^{c\sqrt{\log n}} / n^{8/7})$ for some constant $c > 0$. Recently, for n randomly chosen points in the unit-square $[0, 1]^2$, the expected value of the minimum area of a triangle among these n points was determined to $\Theta(1/n^3)$ by Jiang, Li and Vitány [9].

A variant of Heilbronn's problem considered by Barequet [2] asks, given a fixed integer $d \geq 2$, for the maximum value $\Delta_d^*(n)$ such that there exists a distribution of n points in the d -dimensional unit-cube $[0, 1]^d$ where the minimum volume of a simplex formed by some $(d+1)$ of these n points is equal to $\Delta_d^*(n)$. He showed in [2] the lower bound $\Delta_d^*(n) = \Omega(1/n^d)$, which was improved in [12] to $\Delta_d^*(n) = \Omega(\log n / n^d)$. In [13] a deterministic polynomial time algorithm was given achieving the lower bound $\Delta_3^*(n) = \Omega(\log n / n^3)$. Recently, Brass [6] showed the upper bound $\Delta_d^*(n) = O(1/n^{(2d+1)/(2d)})$ for odd $d \geq 3$, while for even $d \geq 4$ only $\Delta_d^*(n) = O(1/n)$ is known. Moreover, an on-line version of this variant was investigated by Barequet [3] for dimensions $d = 3$ and $d = 4$, where he showed the lower bounds $\Omega(1/n^{10/3})$ and $\Omega(1/n^{127/24})$, respectively.

Here we will investigate the following extension of Heilbronn's problem to higher dimensions: for fixed integers $d \geq 2$ and any given integer $n \geq 3$ find a set of n points in the d -dimensional unit-cube $[0, 1]^d$ such that the minimum area of a triangle determined by three of these n points is maximal. We consider the off-line as well as the on-line version of our problem. In the off-line situation the number n of points is given in advance, while in the on-line case the points are positioned in $[0, 1]^d$ one after the other and at some time this process stops. Let the corresponding maximum values on the minimum triangle areas be denoted by $\Delta_d^{off-line}(n)$ and $\Delta_d^{on-line}(n)$, respectively.

Theorem 1. *Let $d \geq 2$ be a fixed integer. Then, for constants $c_1, c_2, C_1, C_2 > 0$, which depend on d only, for every integer $n \geq 3$ it is*

$$c_1 \cdot \frac{(\log n)^{1/(d-1)}}{n^{2/(d-1)}} \leq \Delta_d^{off-line}(n) \leq \frac{C_1}{n^{2/d}} \quad (1)$$

$$\frac{c_2}{n^{2/(d-1)}} \leq \Delta_d^{on-line}(n) \leq \frac{C_2}{n^{2/d}}. \quad (2)$$

The lower bounds (1) and (2) differ only by a factor of $\Theta((\log n)^{1/(d-1)})$. In contrast to this, the lower bounds in the on-line situation considered by Barequet [3], i.e. maximizing the minimum volume of simplices among n points in $[0, 1]^d$, differ by a factor of $\Theta(n^{1/3} \cdot \log n)$ for dimension $d = 3$ and by a factor of $\Theta(n^{31/24} \cdot \log n)$ for dimension $d = 4$ from the currently best known lower bound $\Delta_d^*(n) = \Omega(\log n/n^d)$ for any fixed integer $d \geq 2$ in the off-line situation.

In the following we will split the statement of Theorem 1 into a few lemmas.

2 The Off-Line Case

A line through points $P_i, P_j \in [0, 1]^d$ is denoted by $P_i P_j$. Let $\text{dist}(P_i, P_j)$ be the Euclidean distance between the points P_i and P_j . The area of a triangle determined by three points $P_i, P_j, P_k \in [0, 1]^d$ is denoted by $\text{area}(P_i, P_j, P_k)$, where $\text{area}(P_i, P_j, P_k) := \text{dist}(P_i, P_j) \cdot h/2$ and h is the Euclidean distance from point P_k to the line $P_i P_j$.

First we will prove the lower bound in (1) from Theorem 1, namely

Lemma 1. *Let $d \geq 2$ be a fixed integer. Then, for some constant $c_1 = c_1(d) > 0$, for every integer $n \geq 3$ it is*

$$\Delta_d^{off-line}(n) \geq c_1 \cdot \frac{(\log n)^{1/(d-1)}}{n^{2/(d-1)}}. \quad (3)$$

Proof. Let $d \geq 2$ be a fixed integer. For arbitrary integers $n \geq 3$ and a constant $\alpha > 0$, which will be specified later, we select uniformly at random and independently of each other $N = n^{1+\alpha}$ points P_1, P_2, \dots, P_N in the d -dimensional unit-cube $[0, 1]^d$. For fixed $i < j < k$ we will estimate the probability that $\text{area}(P_i, P_j, P_k) \leq A$ for some value $A > 0$ which will be specified later. The point P_i can be anywhere in $[0, 1]^d$. Given point P_i , the probability, that point

$P_j \in [0, 1]^d$ has a Euclidean distance from P_i within the infinitesimal range $[r, r + dr]$, is at most the difference of the volumes of the d -dimensional balls with center P_i and with radii $(r + dr)$ and r , respectively. Hence we obtain

$$\text{Prob}(r \leq \text{dist}(P_i, P_j) \leq r + dr) \leq d \cdot C_d \cdot r^{d-1} dr, \quad (4)$$

where throughout this paper C_d is equal to the volume of the d -dimensional unit-ball in \mathbb{R}^d . Given the points P_i and P_j with $\text{dist}(P_i, P_j) = r$, the third point $P_k \in [0, 1]^d$ satisfies $\text{area}(P_i, P_j, P_k) \leq A$, if P_k is contained in the intersection $C_{i,j} \cap [0, 1]^d$ with $C_{i,j}$ being a d -dimensional cylinder, centered at the line $P_i P_j$ with radius $2 \cdot A/r$ and height at most \sqrt{d} . The d -dimensional volume $\text{vol}(C_{i,j} \cap [0, 1]^d)$ is at most $C_{d-1} \cdot (2A/r)^{d-1} \cdot \sqrt{d}$, and we infer for some constant $C'_d > 0$:

$$\begin{aligned} & \text{Prob}(\text{area}(P_i, P_j, P_k) \leq A) \\ & \leq \int_0^{\sqrt{d}} \text{vol}(C_{i,j} \cap [0, 1]^d) \cdot d \cdot C_d \cdot r^{d-1} dr \\ & \leq \int_0^{\sqrt{d}} C_{d-1} \cdot \left(\frac{2 \cdot A}{r}\right)^{d-1} \cdot \sqrt{d} \cdot d \cdot C_d \cdot r^{d-1} dr \\ & = C_{d-1} \cdot C_d \cdot 2^{d-1} \cdot d^{3/2} \cdot A^{d-1} \cdot \int_0^{\sqrt{d}} dr = C'_d \cdot A^{d-1}. \end{aligned} \quad (5)$$

Definition 1. A hypergraph $\mathcal{G} = (V, \mathcal{E})$ with vertex set V and edge set \mathcal{E} is k -uniform if $|E| = k$ for all edges $E \in \mathcal{E}$.

A hypergraph $\mathcal{G} = (V, \mathcal{E})$ is linear if $|E \cap E'| \leq 1$ for all distinct edges $E, E' \in \mathcal{E}$.

A subset $I \subseteq V$ of the vertex set is independent if I contains no edges from \mathcal{E} . The largest size $|I|$ of an independent set in \mathcal{G} is the independence number $\alpha(\mathcal{G})$.

We form a random 3-uniform hypergraph $\mathcal{G} = (V, \mathcal{E}_3)$ with vertex set $V = \{1, 2, \dots, N\}$, where vertex i corresponds to the random point $P_i \in [0, 1]^d$, and with edges $\{i, j, k\} \in \mathcal{E}_3$ if and only if $\text{area}(P_i, P_j, P_k) \leq A$. The expected number $E(|\mathcal{E}_3|)$ of edges satisfies by (5) for some constant $c_d > 0$:

$$E(|\mathcal{E}_3|) \leq \binom{N}{3} \cdot C'_d \cdot A^{d-1} \leq c_d \cdot A^{d-1} \cdot N^3. \quad (6)$$

We will use the following result on the independence number of linear k -uniform hypergraphs due to Ajtai, Komlós, Pintz, Spencer and Szemerédi [1], see also [7], compare Fundia [8] and [4] for deterministic algorithmic versions of Theorem 2:

Theorem 2. [1, 7] Let $k \geq 3$ be a fixed integer. Let $\mathcal{G} = (V, \mathcal{E})$ be a k -uniform hypergraph on $|V| = n$ vertices with average degree $t^{k-1} = k \cdot |\mathcal{E}|/|V|$. If \mathcal{G} is linear, then its independence number $\alpha(\mathcal{G})$ satisfies for some constant $c_k^* > 0$:

$$\alpha(\mathcal{G}) \geq c_k^* \cdot \frac{n}{t} \cdot (\log t)^{\frac{1}{k-1}}. \quad (7)$$

To apply Theorem 2, we estimate in the random hypergraph $\mathcal{G} = (V, \mathcal{E}_3)$ the expected number $E(BP_{D_0}(\mathcal{G}))$ of ‘bad pairs of small triangles’ in \mathcal{G} , which are among the random points $P_1, \dots, P_N \in [0, 1]^d$ those pairs of triangles sharing an edge and both with area at most A , where all sides are of length at least D_0 . Then we will delete one vertex from each ‘pair of points with Euclidean distance at most D_0 ’ and from each ‘bad pair of small triangles’, which yields a linear subhypergraph \mathcal{G}^{**} of \mathcal{G} and \mathcal{G}^{**} fulfills the assumptions of Theorem 2.

Let $P_{D_0}(\mathcal{G})$ be a random variable counting the number of pairs of distinct points with Euclidean distance at most D_0 among the N randomly chosen points. For fixed integers i, j , $1 \leq i < j \leq N$, we have

$$\text{Prob}(\text{dist}(P_i, P_j) \leq D_0) \leq C_d \cdot D_0^d,$$

as point P_i can be anywhere in $[0, 1]^d$ and, given point P_i , the probability that $\text{dist}(P_i, P_j) \leq D_0$ is bounded from above by the volume of the d -dimensional ball with radius D_0 and center P_i . For $D_0 := 1/N^\beta$, where $\beta > 0$ is a constant, the expected number $E(P_{D_0}(\mathcal{G}))$ of pairs of distinct points with Euclidean distance at most D_0 among the N points satisfies for some constant $c'_d > 0$:

$$E(P_{D_0}(\mathcal{G})) \leq \binom{N}{2} \cdot C_d \cdot D_0^d \leq c'_d \cdot N^{2-\beta d}. \quad (8)$$

For distinct points P_i, P_j, P_k, P_l , $1 \leq i < j < k < l \leq N$, there are $\binom{4}{2}$ choices for the joint side of the two triangles, given by the points P_i and P_j , say. Given point P_i , by (4) we have $\text{Prob}(r \leq \text{dist}(P_i, P_j) \leq r + dr) \leq d \cdot C_d \cdot r^{d-1} dr$. Given points P_i and P_j with $\text{dist}(P_i, P_j) = r$, the probability that the triangle formed by P_i, P_j, P_k , or by P_i, P_j, P_l , has area at most A , is at most the volume of the cylinder, which is centered at the line $P_i P_j$ with height \sqrt{d} and radius $2 \cdot A/r$. Thus, for $d \geq 3$ we have for some constant $C''_d > 0$:

$$\begin{aligned} & \text{Prob}(P_i, P_j, P_k, P_l \text{ form a bad pair of small triangles}) \\ & \leq \binom{4}{2} \cdot \int_{D_0}^{\sqrt{d}} \left(C_{d-1} \cdot \sqrt{d} \cdot \left(\frac{2 \cdot A}{r} \right)^{d-1} \right)^2 \cdot d \cdot C_d \cdot r^{d-1} dr \\ & = C''_d \cdot A^{2d-2} \cdot \int_{D_0}^{\sqrt{d}} \frac{dr}{r^{d-1}} \\ & = \frac{C''_d}{d-2} \cdot A^{2d-2} \cdot \left(\frac{1}{D_0^{d-2}} - \frac{1}{d^{(d-2)/2}} \right) \leq C''_d \cdot A^{2d-2} \cdot N^{\beta(d-2)}. \end{aligned} \quad (9)$$

For dimension $d = 2$ the expression (9) is bounded from above by $C''_2 \cdot A^2 \cdot \log N$ for a constant $C''_2 > 0$. Thus, for each $d \geq 2$ we have

$$\begin{aligned} & \text{Prob}(P_i, P_j, P_k, P_l \text{ form a bad pair of small triangles}) \\ & \leq C''_d \cdot A^{2d-2} \cdot N^{\beta(d-2)} \cdot \log N, \end{aligned}$$

and we obtain for the expected number $E(BP_{D_0}(\mathcal{G}))$ of such bad pairs of small triangles among the N points for a constant $c_d'' > 0$ that

$$\begin{aligned} E(BP_{D_0}(\mathcal{G})) &\leq \binom{N}{4} \cdot C_d'' \cdot A^{2d-2} \cdot N^{\beta(d-2)} \cdot \log N \\ &\leq c_d'' \cdot A^{2d-2} \cdot N^{4+\beta(d-2)} \cdot \log N. \end{aligned} \quad (10)$$

Using (6), (8) and (10) and Markov's inequality there exist $N = n^{1+\alpha}$ points in the unit-cube $[0, 1]^d$ such that the corresponding 3-uniform hypergraph $\mathcal{G} = (V, \mathcal{E}_3)$ on $|V| = N$ vertices satisfies

$$|\mathcal{E}_3| \leq 3 \cdot c_d \cdot A^{d-1} \cdot N^3 \quad (11)$$

$$P_{D_0}(\mathcal{G}) \leq 3 \cdot c_d' \cdot N^{2-\beta d} \quad (12)$$

$$BP_{D_0}(\mathcal{G}) \leq 3 \cdot c_d'' \cdot A^{2d-2} \cdot N^{4+\beta(d-2)} \cdot \log N. \quad (13)$$

By (11) the average degree $t^2 = 3 \cdot |\mathcal{E}_3|/|V|$ of $\mathcal{G} = (V, \mathcal{E}_3)$ satisfies $t^2 \leq t_0^2 := 9 \cdot c_d \cdot A^{d-1} \cdot N^3/N = 9 \cdot c_d \cdot A^{d-1} \cdot N^2$. For a suitable constant $\varepsilon > 0$, we pick with probability $p := N^\varepsilon/t_0 \leq 1$ uniformly at random and independently of each other vertices from V . Let $V^* \subseteq V$ be the random set of the picked vertices, and let $\mathcal{G}^* = (V^*, \mathcal{E}_3^*)$ with $\mathcal{E}_3^* = \mathcal{E}_3 \cap [V^*]^3$ be the resulting random induced subhypergraph of \mathcal{G} . Using (11), (12), (13) we infer for the expected numbers of vertices, edges, pairs of points with Euclidean distance at most D_0 and bad pairs of small triangles in \mathcal{G}^* for some constants $c_1, c_2, c_3, c_4 > 0$:

$$\begin{aligned} E(|V^*|) &= p \cdot N \geq c_1 \cdot N^\varepsilon / A^{\frac{d-1}{2}} \\ E(|\mathcal{E}_3^*|) &= p^3 \cdot |\mathcal{E}_3| \leq p^3 \cdot 3 \cdot c_d \cdot A^{d-1} \cdot N^3 \leq c_2 \cdot N^{3\varepsilon} / A^{\frac{d-1}{2}} \\ E(P_{D_0}(\mathcal{G}^*)) &= p^2 \cdot P_{D_0}(\mathcal{G}) \leq p^2 \cdot 3 \cdot c_d' \cdot N^{2-\beta d} \leq c_3 \cdot N^{2\varepsilon-\beta d} / A^{d-1} \\ E(BP_{D_0}(\mathcal{G}^*)) &= p^4 \cdot BP_{D_0}(\mathcal{G}) \leq p^4 \cdot 3 \cdot c_d'' \cdot A^{2d-2} \cdot N^{4+\beta(d-2)} \cdot \log N \leq \\ &\leq c_4 \cdot N^{4\varepsilon+\beta(d-2)} \cdot \log N. \end{aligned}$$

By Chernoff's and Markov's inequality there exists an induced subhypergraph $\mathcal{G}^* = (V^*, \mathcal{E}_3^*)$ of \mathcal{G} such that the following hold:

$$|V^*| \geq (c_1 - o(1)) \cdot N^\varepsilon / A^{\frac{d-1}{2}} \quad (14)$$

$$|\mathcal{E}_3^*| \leq 4 \cdot c_2 \cdot N^{3\varepsilon} / A^{\frac{d-1}{2}} \quad (15)$$

$$P_{D_0}(\mathcal{G}^*) \leq 4 \cdot c_3 \cdot N^{2\varepsilon-\beta d} / A^{d-1} \quad (16)$$

$$BP_{D_0}(\mathcal{G}^*) \leq 4 \cdot c_4 \cdot N^{4\varepsilon+\beta(d-2)} \cdot \log N. \quad (17)$$

Now we fix $\alpha := 1/d$ and $\beta := 1/d$ and for some suitable constant $c^* > 0$ we set

$$A := c^* \cdot \frac{(\log n)^{\frac{1}{d-1}}}{n^{\frac{2}{d-1}}}. \quad (18)$$

Lemma 2. For $0 < \varepsilon \leq (d+2)/(3 \cdot d \cdot (d+1))$ it is

$$BP_{D_0}(\mathcal{G}^*) = o(|V^*|).$$

Proof. Using (14), (17), (18) and $N = n^{1+\alpha}$ we have

$$\begin{aligned}
BP_{D_0}(\mathcal{G}^*) &= o(|V^*|) \\
\iff N^{4\varepsilon+\beta(d-2)} \cdot \log N &= o(N^\varepsilon / A^{\frac{d-1}{2}}) \\
\iff N^{3\varepsilon+\beta(d-2)} \cdot \log N \cdot A^{\frac{d-1}{2}} &= o(1) \\
\iff n^{(1+\alpha)(3\varepsilon+\beta(d-2))-1} \cdot (\log n)^{\frac{1}{2}} &= o(1) \\
\iff (1+\alpha) \cdot (3\varepsilon+\beta(d-2)) &< 1 \\
\iff \varepsilon < \frac{d+2}{3 \cdot d \cdot (d+1)} &\quad \text{as } \alpha = \beta = 1/d. \quad \square
\end{aligned}$$

Lemma 3. For $0 < \varepsilon < 1/(d+1)$ it is

$$P_{D_0}(\mathcal{G}^*) = o(|V^*|).$$

Proof. By (14), (16), (18), using $N = n^{1+\alpha}$, we infer

$$\begin{aligned}
P_{D_0}(\mathcal{G}^*) &= o(|V^*|) \\
\iff N^{2\varepsilon-\beta d} / A^{d-1} &= o(N^\varepsilon / A^{\frac{d-1}{2}}) \\
\iff N^{\varepsilon-\beta d} / A^{\frac{d-1}{2}} &= o(1) \\
\iff n^{(\varepsilon-\beta d)(1+\alpha)+1} / (\log n)^{1/2} &= o(1) \\
\iff (\varepsilon - \beta d)(1+\alpha) &< -1 \\
\iff \varepsilon < \frac{1}{d+1} &\quad \text{as } \alpha = \beta = 1/d. \quad \square
\end{aligned}$$

We fix $\varepsilon := 1/(3d)$, hence $p \leq 1$ for $n > n_0$. In the subhypergraph $\mathcal{G}^* = (V^*, \mathcal{E}_3^*)$ we delete one vertex from each bad pair of small triangles with all side-lengths at least D_0 and from each pair of vertices where the corresponding points have Euclidean distance at most D_0 . The resulting induced subhypergraph $\mathcal{G}^{**} = (V^{**}, \mathcal{E}_3^{**})$ with $\mathcal{E}_3^{**} = [V^{**}]^3 \cap \mathcal{E}_3^*$ fulfills by Lemmas 2 and 3 and by (14), (15):

$$|V^{**}| \geq (c_1 - o(1)) \cdot N^\varepsilon / A^{\frac{d-1}{2}} \quad (19)$$

$$|\mathcal{E}_3^{**}| \leq |\mathcal{E}_3^*| \leq 4 \cdot c_2 \cdot N^{3\varepsilon} / A^{\frac{d-1}{2}} \quad (20)$$

and the points corresponding to the vertices of the subhypergraph \mathcal{G}^{**} do not form any bad pairs of small triangles anymore, i.e. \mathcal{G}^{**} is a linear hypergraph. By (19) and (20) the hypergraph \mathcal{G}^{**} has average degree

$$t^2 \leq t_1^2 := \frac{12 \cdot c_2 \cdot N^{3\varepsilon} / A^{\frac{d-1}{2}}}{(c_1 - o(1)) \cdot N^\varepsilon / A^{\frac{d-1}{2}}} = \frac{(12 + o(1)) \cdot c_2}{c_1} \cdot N^{2\varepsilon}. \quad (21)$$

The assumptions of Theorem 2 are fulfilled by the 3-uniform subhypergraph \mathcal{G}^{**} of \mathcal{G} and we infer for $t \geq 2$ with (7), (19) and (21) for its independence number

$$\begin{aligned}
 \alpha(\mathcal{G}) &\geq \alpha(\mathcal{G}^{**}) \geq c_3^* \cdot \frac{|V^{**}|}{t} \cdot (\log t)^{1/2} \geq c_3^* \cdot \frac{|V^{**}|}{t_1} \cdot (\log t_1)^{1/2} \geq \\
 &\geq \frac{c_3^* \cdot (c_1^{3/2} - o(1)) \cdot N^\varepsilon}{((12 + o(1)) \cdot c_2)^{1/2} \cdot N^\varepsilon \cdot A^{(d-1)/2}} \cdot \left(\log \left(\frac{(12 + o(1)) \cdot c_2}{c_1} \right)^{1/2} \cdot N^\varepsilon \right)^{1/2} \\
 &\geq c' \cdot (\log n)^{1/2} / A^{(d-1)/2} \quad \text{as } N = n^{1+\alpha} \\
 &\geq c' \cdot (1/c^*)^{\frac{d-1}{2}} \cdot \frac{n}{(\log n)^{1/2}} \cdot (\log n)^{1/2} \geq n
 \end{aligned}$$

for some sufficiently small constant $c^* > 0$. Thus the hypergraph \mathcal{G} contains an independent set $I \subseteq V$ with $|I| = n$. These n vertices represent n points in $[0, 1]^d$, where every triangle among these n points has area at least A , i.e. $\Delta_d(n)^{off-line} = \Omega((\log n)^{1/(d-1)} / n^{2/(d-1)})$. \square

3 The On-Line Case

In this section we consider the on-line situation and we will show the lower bound in (2) from Theorem 1:

Lemma 4. *Let $d \geq 2$ be a fixed integer. Then, for some constant $c_2 = c_2(d) > 0$, for every integer $n \geq 3$ it is*

$$\Delta_d^{on-line}(n) \geq \frac{c_2}{n^{2/(d-1)}}.$$

Proof. Successively we will construct an arbitrary long sequence P_1, P_2, \dots of points in the unit-cube $[0, 1]^d$ such that for suitable constants $a, b, \alpha, \beta > 0$, which will be fixed later, for every n the set $S_n = \{P_1, P_2, \dots, P_n\}$ satisfies

- (i) $\text{dist}(P_i, P_j) > a/n^\alpha$ for all $1 \leq i < j \leq n$, and
- (ii) $\text{area}(P_i, P_j, P_k) > b/n^\beta$ for all $1 \leq i < j < k \leq n$.

Assume that a set $S_{n-1} = \{P_1, P_2, \dots, P_{n-1}\} \subset [0, 1]^d$ of $(n-1)$ points with (i') $\text{dist}(P_i, P_j) > a/(n-1)^\alpha$ for all $1 \leq i < j \leq n-1$, and (ii') $\text{area}(P_i, P_j, P_k) > b/(n-1)^\beta$ for all $1 \leq i < j < k \leq n-1$ is already constructed.

To have some space in $[0, 1]^d$ available for choosing a new point $P_n \in [0, 1]^d$ such that (i) is fulfilled, this new point P_n is not allowed to lie within any of the d -dimensional balls $B_r(P_i)$ of radius $r = a/n^\alpha$ with center P_i , $i = 1, 2, \dots, n-1$. Adding the volumes of these balls yields

$$\sum_{i=1}^{n-1} \text{vol}(B_r(P_i)) < n \cdot C_d \cdot r^d = a^d \cdot C_d \cdot n^{1-\alpha d}.$$

For $\alpha := 1/d$ and $a^d \cdot C_d < 1/2$ we have $\sum_{i=1}^{n-1} \text{vol}(B_r(P_i)) < 1/2$.

We will show next that the regions, where condition (ii) is violated, also have volume less than $1/2$. The regions, where (ii) is violated by points $P_n \in [0, 1]^d$, are given by $C_{i,j} \cap [0, 1]^d$, $1 \leq i < j \leq n-1$, where $C_{i,j}$ is a d -dimensional cylinder centered at the line $P_i P_j$. These sets $C_{i,j} \cap [0, 1]^d$ are contained in cylinders of height \sqrt{d} and radius $2 \cdot b / (n^\beta \cdot \text{dist}(P_i, P_j))$. We sum up their volumes:

$$\begin{aligned}
& \sum_{1 \leq i < j \leq n-1} \text{vol}(C_{i,j} \cap [0, 1]^d) \\
& \leq \sum_{1 \leq i < j \leq n-1} \sqrt{d} \cdot C_{d-1} \cdot \left(\frac{2 \cdot b}{n^\beta \cdot \text{dist}(P_i, P_j)} \right)^{d-1} \\
& = \frac{(2 \cdot b)^{d-1} \cdot \sqrt{d} \cdot C_{d-1}}{2 \cdot n^{\beta(d-1)}} \cdot \sum_{i=1}^{n-1} \sum_{j=1; j \neq i}^{n-1} \left(\frac{1}{\text{dist}(P_i, P_j)} \right)^{d-1}. \tag{22}
\end{aligned}$$

We fix some point P_i , $i = 1, 2, \dots, n-1$. To give an upper bound on the last sum, we will use a packing argument, compare [2]. Consider the balls $B_{r_t}(P_i)$ with center P_i and radius $r_t := a \cdot t / n^\alpha$, $t = 0, 1, \dots$ with $t \leq \sqrt{d} \cdot n^\alpha / a$. Clearly $\text{vol}(B_{r_0}(P_i)) = 0$, and for some constant $C_d^* > 0$ for $t = 1, 2, \dots$ we have

$$\text{vol}(B_{r_t}(P_i) \setminus B_{r_{t-1}}(P_i)) \leq C_d^* \cdot \frac{t^{d-1}}{n^{\alpha d}}. \tag{23}$$

Notice that for every ball $B_r(P_j)$ with radius $r = \Theta(n^{-\alpha})$ and center $P_j \in B_{r_t}(P_i) \setminus B_{r_{t-1}}(P_i)$ with $i \neq j$ we have $\text{vol}(B_r(P_j) \cap (B_{r_t}(P_i) \setminus B_{r_{t-1}}(P_i))) = \Theta(n^{-\alpha d})$. By inequalities (i') we have $n_1 = 1$ and by (23) each shell $B_{r_t}(P_i) \setminus B_{r_{t-1}}(P_i)$, $t = 2, 3, \dots$, contains at most $n_t \leq C_d' \cdot t^{d-1}$ points from the set S_{n-1} for some constant $C_d' > 0$. Using the inequality $1 + x \leq e^x$, $x \in \mathbb{R}$, we obtain

$$\begin{aligned}
& \sum_{j=1; j \neq i}^{n-1} \left(\frac{1}{\text{dist}(P_i, P_j)} \right)^{d-1} \leq \sum_{t=2}^{\sqrt{d} \cdot n^\alpha / a} n_t \cdot \left(\frac{1}{a \cdot (t-1) / n^\alpha} \right)^{d-1} \leq \\
& \leq \sum_{t=2}^{\sqrt{d} \cdot n^\alpha / a} C_d' \cdot t^{d-1} \cdot \left(\frac{1}{a \cdot (t-1) / n^\alpha} \right)^{d-1} \leq \sum_{t=2}^{\sqrt{d} \cdot n^\alpha / a} \frac{C_d'}{a^{d-1}} \cdot e^{\frac{d-1}{t-1}} \cdot n^{\alpha(d-1)} \leq \\
& \leq C_d'' \cdot n^{\alpha d} \tag{24}
\end{aligned}$$

for some constant $C_d'' > 0$. We set $\beta := 2/(d-1)$ and, using $\alpha = 1/d$ and (24), inequality (22) becomes for a sufficiently small constant $b > 0$:

$$\begin{aligned}
& \sum_{1 \leq i < j \leq n-1} \text{vol}(C_{i,j} \cap [0, 1]^d) \\
& \leq \frac{(2 \cdot b)^{d-1} \cdot \sqrt{d} \cdot C_{d-1}}{n^{\beta(d-1)}} \cdot \sum_{i=1}^{n-1} \sum_{j=1; j \neq i}^{n-1} \left(\frac{1}{\text{dist}(P_i, P_j)} \right)^{d-1} \\
& \leq \frac{(2 \cdot b)^{d-1} \cdot \sqrt{d} \cdot C_{d-1}}{n^{\beta(d-1)}} \cdot \sum_{i=1}^{n-1} C_d'' \cdot n^{\alpha d} \\
& \leq (2 \cdot b)^{d-1} \cdot \sqrt{d} \cdot C_{d-1} \cdot C_d'' \cdot n^{1+\alpha d - \beta(d-1)} < 1/2.
\end{aligned}$$

The forbidden regions have volume less than 1, hence there exists a point $P_n \in [0, 1]^d$ such that (i) and (ii) are satisfied, thus $\Delta_d^{on-line}(n) = \Omega(1/n^{2/(d-1)})$. \square

4 An Upper Bound

Here we will show the upper bound $O(1/n^{2/d})$ from Theorem 1 on the smallest area of a triangle among any n points in the d -dimensional unit-cube $[0, 1]^d$.

Lemma 5. *Let $d \geq 2$ be a fixed integer. Then, for some constant $C_1 > 0$, for every integer $n \geq 3$ it is*

$$\Delta_d^{on-line}(n) \leq \Delta_d^{off-line}(n) \leq \frac{C_1}{n^{2/d}}.$$

Proof. Given any n points $P_1, P_2, \dots, P_n \in [0, 1]^d$, for some value $D_0 > 0$ we form a graph $G_{D_0} = (V, E)$ with vertex set $V = \{1, 2, \dots, n\}$, where vertex i corresponds to point $P_i \in [0, 1]^d$, and edges $\{i, j\} \in E$ if and only if $\text{dist}(P_i, P_j) \leq D_0$. An independent set $I \subseteq V$ in this graph G_{D_0} yields a subset $I' \subseteq \{P_1, P_2, \dots, P_n\}$ of points with Euclidean distance between any two distinct points bigger than D_0 . Each ball $B_r(P_j)$ with center $P_j \in [0, 1]^d$ and radius $r \leq 1$ satisfies $\text{vol}(B_r(P_j) \cap [0, 1]^d) \geq \text{vol}(B_r(P_j))/2^d$. The balls with radius $D_0/2$ and centers from the set I' have pairwise empty intersection, thus

$$\alpha(G_{D_0}) \cdot 2^{-d} \cdot C_d \cdot (D_0/2)^d \leq \text{vol}([0, 1]^d) = 1. \quad (25)$$

By Turán's theorem [20], for any graph $G = (V, E)$ we have the lower bound $\alpha(G) \geq n/(2 \cdot t)$ on the independence number $\alpha(G)$, where $t := 2 \cdot |E|/|V|$ is the average degree of G . This with (25) implies

$$\frac{4^d}{C_d \cdot D_0^d} \geq \alpha(G_{D_0}) \geq \frac{n}{2 \cdot t} \implies t \geq \frac{C_d}{2 \cdot 4^d} \cdot n \cdot D_0^d.$$

Let $D_0 := c/n^{1/d}$ where $c > 0$ is a constant with $c^d > 2 \cdot 4^d/C_d$. Then $t > 1$ and there exist two edges $\{i, j\}, \{i, k\} \in E$ incident at vertex $i \in V$. Then the two points P_j and P_k have Euclidean distance at most D_0 from point P_i , and hence $\text{area}(P_i, P_j, P_k) \leq D_0^2/2 = 1/2 \cdot c^2/n^{2/d}$, i.e. $\Delta_d^{off-line}(n) = O(1/n^{2/d})$. \square

5 Conclusion

Certainly it is of interest to improve the bounds given in this paper. Also, for the off-line case it is desirable to get a deterministic polynomial time algorithm achieving the bound $\Delta_d(n)^{off-line} = \Omega((\log n)^{1/(d-1)}/n^{2/(d-1)})$. In view of the results in [9] it is also of interest to determine the expected value of the minimum triangle area with respect to the uniform distribution of n points in $[0, 1]^d$.

References

1. M. Ajtai, J. Komlós, J. Pintz, J. Spencer and E. Szemerédi, *Extremal Uncrowded Hypergraphs*, Journal of Combinatorial Theory Ser. A, 32, 1982, 321–335.
2. G. Barequet, *A Lower Bound for Heilbronn’s Triangle Problem in d Dimensions*, SIAM Journal on Discrete Mathematics 14, 2001, 230–236.
3. G. Barequet, *The On-Line Heilbronn’s Triangle Problem in Three and Four Dimensions*, Proceedings ‘8rd Annual International Computing and Combinatorics Conference COCOON’02’, LNCS 2387, Springer, 2002, 360–369.
4. C. Bertram-Kretzberg and H. Lefmann, *The Algorithmic Aspects of Uncrowded Hypergraphs*, SIAM Journal on Computing 29, 1999, 201–230.
5. C. Bertram-Kretzberg, T. Hofmeister and H. Lefmann, *An Algorithm for Heilbronn’s Problem*, SIAM Journal on Computing 30, 2000, 383–390.
6. P. Brass, *An Upper Bound for the d -Dimensional Heilbronn Triangle Problem*, preprint, 2003.
7. R. A. Duke, H. Lefmann and V. Rödl, *On Uncrowded Hypergraphs*, Random Structures & Algorithms 6, 1995, 209–212.
8. A. Fundia, *Derandomizing Chebychev’s Inequality to find Independent Sets in Uncrowded Hypergraphs*, Random Structures & Algorithms, 8, 1996, 131–147.
9. T. Jiang, M. Li and P. Vitany, *The Average Case Area of Heilbronn-type Triangles*, Random Structures & Algorithms 20, 2002, 206–219.
10. J. Komlós, J. Pintz and E. Szemerédi, *On Heilbronn’s Triangle Problem*, Journal of the London Mathematical Society, 24, 1981, 385–396.
11. J. Komlós, J. Pintz and E. Szemerédi, *A Lower Bound for Heilbronn’s Problem*, Journal of the London Mathematical Society, 25, 1982, 13–24.
12. H. Lefmann, *On Heilbronn’s Problem in Higher Dimension*, Combinatorica 23, 2003, 669–680.
13. H. Lefmann and N. Schmitt, *A Deterministic Polynomial Time Algorithm for Heilbronn’s Problem in Three Dimensions*, SIAM Journal on Computing 31, 2002, 1926–1947.
14. K. F. Roth, *On a Problem of Heilbronn*, Journal of the London Mathematical Society 26, 1951, 198–204.
15. K. F. Roth, *On a Problem of Heilbronn, II*, Proc. of the London Mathematical Society (3), 25, 1972, 193–212.
16. K. F. Roth, *On a Problem of Heilbronn, III*, Proc. of the London Mathematical Society (3), 25, 1972, 543–549.
17. K. F. Roth, *Estimation of the Area of the Smallest Triangle Obtained by Selecting Three out of n Points in a Disc of Unit Area*, Proc. of Symposia in Pure Mathematics, 24, 1973, AMS, Providence, 251–262.
18. K. F. Roth, *Developments in Heilbronn’s Triangle Problem*, Advances in Mathematics, 22, 1976, 364–385.
19. W. M. Schmidt, *On a Problem of Heilbronn*, Journal of the London Mathematical Society (2), 4, 1972, 545–550.
20. P. Turán, *On an Extremal Problem in Graph Theory*, Mat. Fiz. Lapok 48, 1941, 436–452.

Progress on Maximum Weight Triangulation

Francis Y.L. Chin¹, Jianbo Qian², and Cao An Wang³

¹ Department of Computer Science and Information Systems,
The University of Hong Kong, Hong Kong
`chin@csis.hku.hk`

² Institute of Applied Mathematics,
Chinese Academy of Science, Beijing, China
`jbqian@mail.amss.ac.cn`

³ Department of Computer Science, Memorial University of Newfoundland,
St. John's, Newfoundland, Canada A1B 3X5
`wang@garfield.cs.mun.ca`

Abstract. In this paper, we investigate the maximum weight triangulation of a point set in the plane. We prove that the weight of maximum weight triangulation of any planar point set with diameter D is bounded above by $((2\epsilon+2) \cdot n + \frac{\pi(1-2\epsilon)}{8\epsilon\sqrt{1-\epsilon^2}} + \frac{\pi}{2} - 5(\epsilon+1))D$, where ϵ for $0 < \epsilon \leq \frac{1}{2}$ is a constant and n is the number of points in the set. If we use ‘spoke-scan’ algorithm to find a triangulation of the point set, we obtain an approximation ratio of 4.238. Furthermore, if the point set forms a ‘semi-lune’ or a ‘semi-circled’ convex polygon, then its maximum weight triangulation can be found in $O(n^2)$ time.

Keywords: Algorithm, Approximation, Maximum weight triangulation.

1 Introduction

Triangulation of a set of points is a fundamental structure in computational geometry. Among different triangulations, the *minimum weight triangulation* (*MWT* for short) of a set of points in the plane attracts special attention [3,5,6,7]. The construction of the *MWT* of a point set is still an outstanding open problem. When the given point set is the set of vertices of a convex polygon (so-called *convex point set*), then the corresponding *MWT* can be found in $O(n^3)$ time by dynamic programming [3,5]. There are many approximation algorithms for various *MWT*s. A constant ratio of *MWT* of a point set in the plane was proposed in [7]. However, the constant is so large that its exact value is not known.

On the contrast, there is not much research done on *maximum weight triangulation* (*MAT* for short). From the theoretical viewpoint, the maximum weight triangulation problem and the minimum weight triangulation problem attracts equally interest, and one seems not to be easier than the other. The study of maximum weight triangulation will help us to understand the nature of optimal triangulations.

The first work in the *MAT* [9] showed that if an n -sided polygon P inscribed on a circle, then $MAT(P)$ can be found in $O(n^2)$ time. Recently, a factor 6 approximation algorithm to find the *MAT* of a set of points in the plane was proposed in [4].

In this paper, we present in Section 3 a non-trivial upper-bound of the weight of $MAT(S)$ for a set S of n points in the plane. That is, $\omega(MAT(S)) \leq ((2 + 2\epsilon) \cdot n + \frac{\pi(1-2\epsilon)}{8\epsilon \sqrt{1-\epsilon^2}} + \frac{\pi}{2} - 5(\epsilon + 1))D_S$, where D_S is the diameter of S and ϵ is any constant such that $0 < \epsilon \leq \frac{1}{2}$. We also show that when $\epsilon \rightarrow 0$, the upper-bound is tight.

Applying a simple ‘spoke-scan’ approximation algorithm in [4], we obtain a triangulation $T(S)$ with ratio $(\frac{\omega(MAT(S))}{\omega(T(S))} =) 4.238$, which improves the previous ratio of 6. Furthermore, let P be a special convex n -sided polygon such that the entire polygon P is contained inside the circle with an edge of P as diameter. We call such a polygon as *semi-circled*. In Section 4, we propose an $O(n^2)$ algorithm for computing the $MAT(P)$ of a semi-circled convex polygon P . Recall that a straightforward dynamic programming method will take $O(n^3)$ to find $MAT(P)$. This algorithm works for more general so-called ‘semi-lune’ convex polygon.

1.1 Preliminaries

Let S be a set of points in the plane. A *triangulation* of S , denoted by $T(S)$, is a maximal set of non-crossing line segments with their endpoints in S . It follows that the interior of the convex hull of S is partitioned into non-overlapping triangles. The weight of a triangulation $T(S)$ is given by

$$\omega(T(S)) = \sum_{\overline{s_i s_j} \in T(S)} \omega(\overline{s_i s_j}),$$

where $\omega(\overline{s_i s_j})$ is the Euclidean length of line segment $\overline{s_i s_j}$.

A *maximum weight triangulation* of S ($MAT(S)$) is defined as for all possible $T(S)$, $\omega(MAT(S)) = \max\{\omega(T(S))\}$.

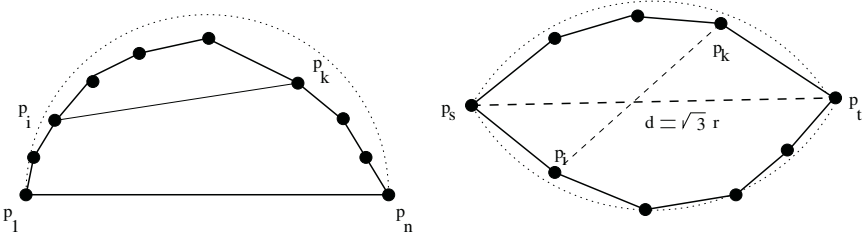


Fig. 1. An illustration of semi-lune and semi-circled convex polygons.

Let P be a convex polygon (whose vertices form a *convex point set*) and $T(P)$ be its triangulation. A *semi-lune convex polygon* P is a special convex polygon

such that its diameter is the diameter of a regular lune and it lies entirely in the lune. (A regular lune is the intersection of two circles passing each others' centers, i.e., the diameter of a regular lune is $\sqrt{3} \cdot r$ for radius r .) A *semi-circled convex polygon* P is a special convex polygon such that its longest edge is the diameter and all the edges of P lie inside the semi-circle with this longest edge as diameter. (Refer to Figure 1).

The following two properties are easy to verify for a semi-circled convex polygon $P = (p_1, p_2, \dots, p_k, \dots, p_{n-1}, p_n)$.

Property 1: Let $\overline{p_i p_k}$ for $1 \leq i < k \leq n$ be an edge in P . Then the area bounded by $\overline{p_i p_k}$ and chain (p_i, \dots, p_k) is a semi-circled convex polygon.

Property 2: In P , edge $\overline{p_i p_j}$ for $1 \leq i < j < n$ or $1 < i < j \leq n$ is shorter than $\overline{p_1 p_n}$. The 'spoke-scan' approximation algorithm proposed in [4] can be described as follows. Let $D_S = \overline{s_i s_j}$ be the diameter of a point set S . The line extending D_S partition S into S^1 and S^2 (excluding s_i and s_j). Let $\omega(E_i^1)$ and $\omega(E_j^1)$ (respectively, $\omega(E_i^2)$ and $\omega(E_j^2)$) denote the sum of the lengths of edges connecting every points in S^1 to s_i and to s_j , respectively. These edges are called 'spokes'. The algorithm first takes the spokes in the larger one of $\omega(E_i^1)$ and $\omega(E_j^1)$ (respectively, the larger one of $\omega(E_i^2)$ and $\omega(E_j^2)$) as the first subset of the triangulation edges. Then, it uses Graham's scan algorithm to complete the triangulation. It is shown that this triangulation has weight at least $(n+1)\frac{D_S}{2}$. We shall use this algorithm in our paper.

Property 3: The diameter of semi-lune convex polygon P belongs to the $MAT(P)$.

Proof: A brief idea of the proof is as follows. Suppose for contradiction that a $MAT(P)$ contains edge $p_i p_k$ which crosses diameter $p_s p_t$ (refer to Figure 1). Note in a regular lune that the longest edge in any triangle spanning on the boundary of and p_s (or p_t) must be incident at p_s (or p_t). Then, both the longer edge of $p_s p_i$ and $p_t p_i$ and the longer edge of $p_s p_k$ and $p_t p_k$ must be longer than or equal to $p_i p_k$. Then, we can construct a new triangulation $T(P)$ including diameter $p_s p_t$ and the weight of $T(P)$ would be large than that of $MAT(P)$. \square

2 Tight Upper-Bound for Maximum-Weight Triangulations

The intuitive idea of proof is as follows. The area of a point set is bounded above by its diameter D (i.e., $\frac{\pi}{4}D^2$). In order to maximize the edge lengths of a triangulation of the set, one may wish that a triangulation contains as many long and thin (called 'large') triangles as possible. However, the area of a triangle and its perimeter are related (by Heron's formula). Thus, the number of large triangles with perimeter greater than $2D$ is limited above by the area of the point set. Consequently, we can bound the total edge-length of any MAT of the set through the estimation of the number of large and the remaining triangles. (Note that constant ϵ is functioned as the shape parameter of a triangle.)

Let D_S denote the diameter of a finite point set S in the plane, let A_S denote the area bounded by the convex hull $CH(S)$, and let L_S denote the perimeter of $CH(S)$. Let constant ϵ be in range $0 < \epsilon \leq \frac{1}{2}$.

It is known in [1,2] that the values of A_S and L_S are bounded above by $\frac{\pi}{4}D_S^2$ and πD_S , respectively.

The following lemma demonstrates the relationship among the area of a triangle, A_Δ , the perimeter of the triangle, $\omega(\Delta)$, and the diameter of the triangle, D_Δ .

Lemma 1. *For a given constant ϵ : $0 < \epsilon \leq \frac{1}{2}$, if perimeter $\omega(\Delta)$ satisfies: $\omega(\Delta) \geq (2 + 2\epsilon)D_\Delta$, then the area $A_\Delta \geq \epsilon\sqrt{1 - \epsilon^2} \cdot D_\Delta^2$.*

Proof: Let a, b , and c be the lengths of the three edges of a triangle Δ . We shall first prove that if A_Δ is minimized, then two of a, b , and c equal D_Δ , and the third one equals 2ϵ .

Note that, when $\omega(\Delta) \geq (2 + 2\epsilon)D_\Delta$ and A_Δ is minimized, the equality:

$$a + b + c = (2 + 2\epsilon)D_\Delta \quad (1)$$

must hold. Otherwise (i.e., $a + b + c > (2 + 2\epsilon)D_\Delta$), we can always “shrink” the triangle to reduce its area while maintaining $\omega(\Delta) \geq (2 + 2\epsilon)D_\Delta$ and without reducing D_Δ until the equality (1) holds.

Let $l = \frac{a+b+c}{2}$. By Heron’s formula, $A_\Delta = \sqrt{l(l-a)(l-b)(l-c)}$.

Without loss of generality, we fix $a = D_\Delta$. We claim that if A_Δ is minimized, then b or c is maximized, that is, one of b and c equals D_Δ . In fact, the following equality holds.

$$(l-b)(l-c) = \frac{(((l-b)+(l-c))^2 - ((l-b)-(l-c))^2)}{4} = \frac{((2l-(b+c))^2 - (b-c)^2)}{4}.$$

Then,

$$A_\Delta = \sqrt{l(l-a) \cdot \frac{((2l-(b+c))^2 - (b-c)^2)}{4}}. \quad (2)$$

Since l and a are fixed, $(b+c)$ is also fixed. So when A_Δ is minimized, $(b-c)^2$ in (2) must be maximized, that is, b or c must equal D_Δ .

Therefore, when A_Δ is minimized, two of a, b , and c equal $D_{triangle}$, and by $a + b + c = (2 + 2\epsilon)D_\Delta$, the third one equals $2\epsilon D_{triangle}$.

By (1) and definition of l , $l = (1 + \epsilon) \cdot D_\Delta$. Without loss of generality, let $b = D_\Delta$, we have $A_\Delta = \sqrt{(1 + \epsilon)D_\Delta \cdot \epsilon D_\Delta \cdot (1 - \epsilon)D_\Delta} = \epsilon\sqrt{1 - \epsilon^2}D_\Delta^2$. We now have $A_\Delta \geq \epsilon\sqrt{1 - \epsilon^2}D_\Delta^2$ for $0 < \epsilon \leq \frac{1}{2}$. \square

In a triangulation of S , $T(S)$, we call a triangle t ‘large’ if the perimeter of t , $\omega(t)$, is at least $(2 + 2\epsilon) \cdot D_S$; and t is ‘small’, otherwise. By Lemma 1 and by the fact that the maximum area of the convex hull of a point set with diameter D_S is bounded above by $\frac{\pi}{4}D_S^2$ [1,2], and note that D_Δ is at most D_S , we have an upper-bound for the number of large triangles in $T(S)$.

Lemma 2. *$T(S)$ may contain at most $\frac{\pi}{4\epsilon\sqrt{1-\epsilon^2}}$ large triangles.*

Now we can derive the upper-bound of the length, $\omega(T)$, of $T(S)$:

Theorem 1. $\omega(T) \leq ((2 + 2\epsilon) \cdot n + \frac{\pi(1-2\epsilon)}{8\epsilon\sqrt{1-\epsilon^2}} + \frac{\pi}{2} - 5(1 + \epsilon))D_S$.

Proof: Let m denote the number of large triangles. It is well known that $T(S)$ contains at most $2n - 5$ triangles. Thus, $T(S)$ has at most $2n - 5 - m$ small triangles. It is clear that the perimeter of a large triangle is at most $3D_S$, and the perimeter of a small one is less than $(2 + 2\epsilon) \cdot D_S$ by definition. If we add up the perimeters of all the triangles in $T(S)$ as well as the perimeter of $CH(S)$, L_S , we obtain a sum of the lengths equals twice of $\omega(T)$. Thus,

$$\begin{aligned} \omega(T) &\leq \frac{1}{2}((2 + 2\epsilon)(2n - 5 - m)D_S + 3mD_S + L_S) \\ &\leq \frac{1}{2}((2 + 2\epsilon)(2n - 5 - \frac{\pi}{4\epsilon\sqrt{1-\epsilon^2}}) + 3\frac{\pi}{4\epsilon\sqrt{1-\epsilon^2}} + \pi)D_S \\ &\quad (\text{note that } m \leq \frac{\pi}{4\epsilon\sqrt{1-\epsilon^2}}, 0 < \epsilon \leq \frac{1}{2}, \text{ and } L_S \leq \pi D_S) \\ &= ((2 + 2\epsilon)D_S \cdot n + (\frac{\pi(1-2\epsilon)}{8\epsilon\sqrt{1-\epsilon^2}} + \frac{\pi}{2} - 5(1 + \epsilon))D_S). \end{aligned}$$

□

Remark 1: Note that the above upper-bound depends on parameter ϵ . When $\epsilon = \frac{1}{2}$, our bound is very close to an obvious upper-bound, $3n - 6$ for $D_S = 1$. On the other hand, when ϵ is sufficiently small, the upper-bound in Theorem 1 is the best possible one with the measurement of D_S . This is because there exists a point set whose MAT has weight $(2 - \frac{2\sqrt{3}\delta}{D_S})D_S \cdot n - 3(D_S - (\sqrt{3} + 1)\delta)$ for any constant $\delta > 0$. When δ is sufficiently small, the weight is approaching to the upper-bound.

In more detail for the claim above, let us consider the following point set.

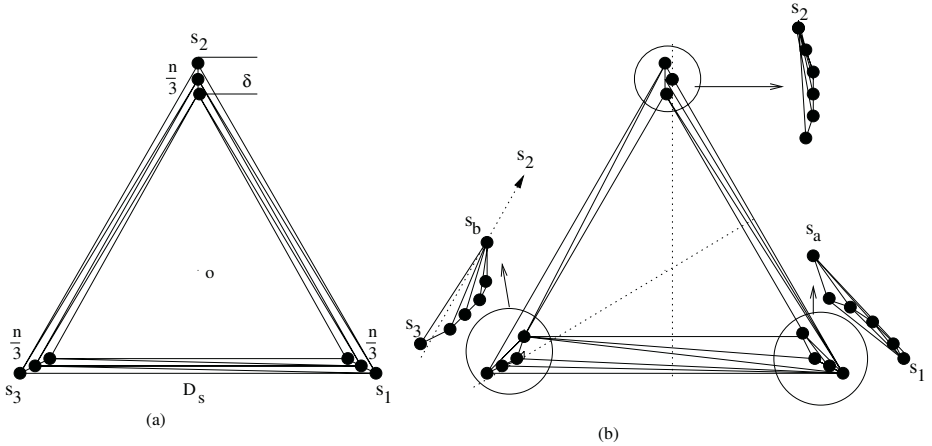


Fig. 2. An illustration of the tight upper-bound (part(a)), and a bad case for ‘spoke-scan’ algorithm (part (b)).

We start to place three points in the vertices of a equilateral triangle, say s_1, s_2 , and s_3 . Let o be the center of the circle inscribing at $\triangle s_1 s_2 s_3$. Now, we place a group of $\frac{n}{3} - 1$ points evenly on line segment $\overline{os_1}$ such that the distance from s_1 to the farthest point in this group is δ . Similarly, we place $\frac{n}{3} - 1$ points

on line segment $\overline{os_2}$ and $\frac{n}{3} - 1$ points on line segment $\overline{os_3}$ by the same manner. (Refer to part(a) of Figure 2.) To construct a triangulation, we connect all points in the neighboring groups by edges. Thus, we obtain a triangulation T' with $n - 3$ short edges and $2n - 3$ long edges, where the sum of the lengths of the shorter edges is no less than 3δ and the sum of the lengths of the long edges is large than $(2n - 3) \cdot (D_S - \sqrt{3}\delta)$. Then, the total length of the triangulation, $\omega(T')$, is at least $(2n - 3)(D_S - \sqrt{3}\delta) + 3\delta$. Then, $\omega(T') \geq 2nD_S - 2n\sqrt{3}\delta - 3D_S + 3\sqrt{3}\delta + 3\delta = (2 - \frac{2\sqrt{3}\delta}{D_S})D_S \cdot n - 3(D_S - (\sqrt{3} + 1)\delta)$. \square

Recall that the approximation ‘Spoke Algorithm’ produces a triangulation with length at least $(n + 1) \cdot \frac{D_S}{2}$.

If we apply this spoke algorithm to produce a triangulation of any point set, then by our upper-bound, we obtain a performance ratio as stated in the following theorem.

Theorem 2. $\frac{\omega(T(S))}{\omega(T'(S))} \rightarrow_{n \rightarrow \infty} 4 + 4\epsilon$ for $0 < \epsilon \leq \frac{1}{2}$, where $T'(S)$ is produced by ‘spoke-scan’ approximation algorithm and $\omega(T(S))$ is the upper-bound in Theorem 1.

Proof: $\frac{\omega(T(S))}{\omega(T'(S))} = \frac{2n(1+\epsilon)D_S + (\frac{\pi(1-2\epsilon)}{8\epsilon\sqrt{1-\epsilon^2}} + \frac{\pi}{2} - 5(1+\epsilon)) \cdot D_S}{(n+1) \cdot \frac{D_S}{2}} = (4 + 4\epsilon) + \frac{b(\epsilon)}{n+1}$, where $b(\epsilon) = \frac{\pi(1-2\epsilon)}{4\epsilon\sqrt{1-\epsilon^2}} + \pi - 14(1+\epsilon)$. When n is sufficiently large, we obtain the desired ratio. \square

Remark 2: When the value of ϵ is properly chosen, the $4 + 4\epsilon$ ratio will apply to any n . To see this, let $\frac{\omega(T(S))}{\omega(T'(S))} \leq 4 + 4\epsilon$, that is, $\frac{2n(1+\epsilon)D_S + (\frac{\pi(1-2\epsilon)}{8\epsilon\sqrt{1-\epsilon^2}} + \frac{\pi}{2} - 5(1+\epsilon)) \cdot D_S}{(n+1) \cdot \frac{D_S}{2}} \leq (4 + 4\epsilon)$ for any $n > 0$. We have that $2(\frac{\pi(1-2\epsilon)}{8\epsilon\sqrt{1-\epsilon^2}} + \frac{\pi}{2} - 5(1+\epsilon)) \leq 4 + 4\epsilon$ after simplification. Solving this inequality for ϵ , we have $\epsilon \geq .05932390326$.

Therefore, when we apply ‘spoke-scan’ algorithm to any point set with $n \geq 3$, we have an approximation ratio of 4.238. \square

3 Finding MAT of Semi-lune and Semi-circled Convex Polygons

We first shall show an $O(n^2)$ dynamic programming algorithm for finding the $MAT(P)$ of semi-circled convex polygon P with n vertices.

Lemma 3. Let $P = (p_1, p_2, \dots, p_k, \dots, p_{n-1}, p_n)$ be a semi-circled convex polygon. Then, one of the edges: $\overline{p_1 p_{n-1}}$ and $\overline{p_n p_2}$ must belong to its maximum weight triangulation $MAT(P)$.

Proof: Suppose for contradiction that none of the two extreme edges: $\overline{p_1 p_{n-1}}$ and $\overline{p_n p_2}$ belongs to $MAT(P)$. Then, there must exist a vertex p_k for $2 < k < n - 1$ such that both $\overline{p_1 p_k}$ and $\overline{p_n p_k}$ belong to $MAT(P)$. Without loss of generality, let p_k lie on one side of the perpendicular bisector of edge $\overline{p_1 p_n}$, say the lefthand side (if p_k is on the bisector, the following argument is still applicable), the two

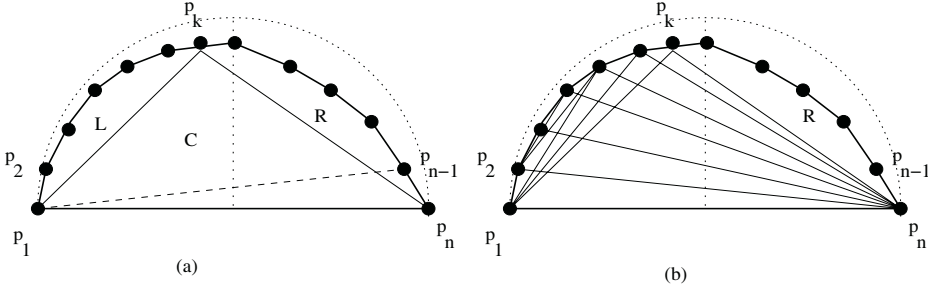


Fig. 3.

edges $\overline{p_1 p_k}$ and $\overline{p_n p_k}$ partition P into three areas, denoted by R , L , and C . Both L and R are semi-circled convex polygons by Property 1. (Refer to part (a) of Figure 3.)

Let the edges of $MAT(P)$ lying inside area L be $(e_1, e_2, \dots, e_{k-3})$ and let $E_L = (e_1, e_2, \dots, e_{k-3}, \overline{p_1 p_k})$. Let E_L^* denote the edges: $(\overline{p_n p_2}, \overline{p_n p_3}, \dots, \overline{p_n p_{k-1}})$. Then, there is a perfect matching between E_L and E_L^* . This is because E_L and E_L^* respectively triangulate the same area $L \cup C$, hence the number of internal edges of the two triangulations must be equal. Let us consider a pair in the matching $(e_i, \overline{p_n p_j})$ for $1 < i, j < k$. It is not hard to see that $\omega(e_i) < \omega(\overline{p_n p_j})$ because any edge in E_L is shorter than $\overline{p_1 p_k}$ by Property 2, any edge in E_L^* is longer than $\overline{p_n p_k}$, and $\overline{p_n p_k}$ is longer than $\overline{p_1 p_k}$ (due to p_k lying on the lefthand side of the perpendicular bisector of $\overline{p_1 p_n}$). Therefore, $\omega(E_L)$ is less than $\omega(E_L^*)$. Now, we shall construct a new triangulation, say $T(P)$, which consists of all the edges in $MAT(P)$ except replacing the edges of E_L by E_L^* . We have that $\omega(T(P)) > \omega(MAT(P))$, which contradicts the $MAT(P)$ assumption. Then, such a p_k cannot exist and one of $\overline{p_1 p_{n-1}}$ and $\overline{p_n p_2}$ must belong to $MAT(P)$. \square

By Lemma 3 and by Property 1, we have a recurrence for the weight of a $MAT(P)$. Let $\omega(i, j)$ denote the weight of the $MAT(P_{i,j})$ of a semi-circled convex polygon $P_{i,j} = (p_i, p_{i+1}, \dots, p_j)$. Let $\omega(\overline{i, j})$ denote the length of edge $\overline{p_i p_j}$.

$$\omega(i, j) = \begin{cases} \omega(\overline{i, i+1}) & j = (i+1) \\ \max\{\omega(i, j-1) + \omega(\overline{j-1, j}), \omega(i+1, j) + \omega(\overline{i, i+1})\} + \omega(\overline{i, j}) & \text{otherwise} \end{cases}$$

It is a straight-forward matter to design a dynamic programming algorithm for finding the $MAT(P)$.

ALGORITHM $MAT - FIND(P)$

Input: a semicircled convex n -sided polygon: (p_1, \dots, p_n) .

Output: $MAT(P)$.

Method:

1. **for** $i = 1$ **to** $n - 1$ **do**
 $\omega(i, i + 1) = \omega(\overline{i, i + 1})$
2. **for** $l = 2$ **to** $n - 1$ **do**
3. **for** $i = 1$ **to** $n - l$ **do**
 $\omega(i, i + l) = \max\{\omega(i, i + l - 1) + \omega(\overline{i + l - 1, i + l}),$
 $\omega(i + 1, i + l) + \omega(\overline{i, i + 1})\} + \omega(\overline{i, i + l})$
4. Identify the edges of $MAT(P)$ by checking the ω 's.
5. **end.**

Since the loop indices i and l range roughly from 1 to n and each evaluation of $\omega(i, j)$ takes constant time, all $\omega(i, j)$ for $1 \leq i, j \leq n$ can be evaluated in $O(n^2)$ time. If we record these ω 's, we can find the edges in $MAT(P)$ by an extra $O(n)$ time to examine the record.

Therefore, we have the following theorem.

Theorem 3. *The maximum weight triangulation of a semi-circled convex n -gon P , $MAT(P)$, can be found in $O(n^2)$ time.*

Proof: The correctness is due to Property 1. It is clear that the number of executions of Step 3 dominates the time complexity. The number of executions is $(n - 3) + (n - 4) + \dots + 2 + 1 \in O(n^2)$. \square

By Property 3 and Theorem 3, we have that

Corollary 1. *The maximum weight triangulation of a semi-lune convex n -gon P , $MAT(P)$, can be found in $O(n^2)$ time.*

4 Conclusion

In this paper, we derived a non-trivial tight upper-bound of the maximum weight triangulation of a point set in the plane. We also used ‘spoke-scan’ algorithm to obtain a factor 4.238 triangulation for any point set in the plane, which improved the previous result: factor 6. We finally proposed an $O(n^2)$ dynamic programming algorithm for constructing the $MAT(P)$ of a semi-lune or a semi-circled convex n -sided polygon.

It is interesting to see what is the range of the performance ratios of the spok e-scan algorithm. There is a good case in which the algorithm produces an optimal solution of $(\frac{D}{2} + \delta)(n + 1)$ for the following point set.

Given any constant $\delta > 0$, we place n points on the unit interval. Two of them locate at the ends of the interval and the rest are in interval $(\frac{1}{2} - \delta, \frac{1}{2} + \delta)$. Then raise the points near the center slightly so that the n points form a convex polygon, and every point near the center is at most $\frac{1}{2} + \delta$ to any of the end points. (Refer to Figure 4.) It is not difficult to see that any triangulation of these n points has length at most $(\frac{D}{2} + \delta)(n + 1)$.

There is a bad case in which the spoke-scan algorithm produces a solution with ratio 3 (refer to part (b) of Figure 2). For the point set in part (a) of

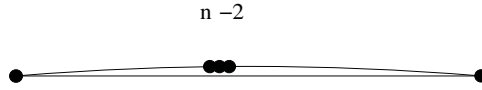


Fig. 4. An illustration of the good case.

Figure 2, we slightly perturb it so that the sum of the lengths of spokes from s_1 is the largest among all the cases. We then arrange each group of $\frac{n}{3}$ points into a convex position. We finally move points s_a and s_b into positions so that during the scan stage, there is no long edge created from the internal points of these groups on the convex side. It is not difficult to see that the optimal solution is $(2n - 3)(D_S - \sqrt{3}\delta) + 3\delta$ and the spoke-scan algorithm produces a solution of $\frac{2}{3}nD_S + 4D_S + n\delta$. The ratio is approaching to 3 for sufficiently small δ and large n .

It is still an open problem whether one can design an $o(n^3)$ algorithm for finding the $MAT(P)$ for a general convex n -sided polygon P .

Acknowledgement

This work is supported by RGC grant HKU 7142/03E for F. Chin and NSERC grant OPG0041629 for C. Wang.

References

1. Scott P.R. and Awyong P.W., Inequalities for convex sets, *Journal of Inequalities in Pure and Applied Mathematics*, Volume 1, Issue 1, Article 6, 2000.
2. Yaglom I.M. and Boltyanskii V.G., *Convex figures*, (Translated by P.J.Kelly and L.F. Walton), Holt, Rinehart and Winston, New York, 1961.
3. Gilbert P., New results on planar triangulations, Tech. Rep. ACT-15 (1979), Coord. Sci. Lab., University of Illinois at Urbana.
4. Hu S., A constant-factor approximation for Maximum Weight triangulation, *Proceeding of 15th Canadian Conference of Computational Geometry*, Halifax, NS Canada, pp.150-154.
5. Klinecsek G., Minimal triangulations of polygonal domains, *Annual Discrete Mathematics* 9 (1980) pp. 121-123.
6. Levcopoulos C. and Lingas A., On approximation behavior of the greedy triangulation for convex polygon, *Algorithmica* 2 (1987), pp. 175-193.
7. Levcopoulos C. and Krznaric D., Quasi-Greedy Triangulations Approximating the Minimum Weight Triangulation. In *Proceeding of the 7th ACM-SIAM Symposium on Discrete Algorithms(SODA)*, pages 392-401, 1996.
8. Preparata F. and Shamos M., *Computational Geometry* (1985), Springer-Verlag.
9. Wang C., Chin F., and Yang B., Maximum Weight triangulation and graph drawing, *The Information Processing Letters*, 70(1999) pp. 17-22.

Coloring Octrees[★]

Udo Adamy¹, Michael Hoffmann¹, József Solymosi², and Miloš Stojaković^{1,★★}

¹ Institute of Theoretical Computer Science, ETH Zurich, CH-8092 Zurich, Switzerland
{adamy, hoffmann, smilos}@inf.ethz.ch

² Department of Mathematics, University of British Columbia, Vancouver, Canada
solymosi@math.ubc.ca

Abstract. An octree is a recursive partition of the unit cube, such that in each step a cube is subdivided into eight smaller cubes. Those cubes that are not further subdivided are the leaves of the octree. We consider the problem of coloring the leaves of an octree using as few colors as possible such that no two of them get the same color if they share a face. It turns out that the number of colors needed depends on a parameter that we call unbalancedness. Roughly speaking, this parameter measures how much adjacent cubes differ in size. For most values of this parameter we give tight bounds on the minimum number of colors, and extend the results to higher dimensions.

1 Introduction

Octrees are a fundamental data structure to store any three-dimensional data in such a way that search and insert operations can be performed efficiently. Therefore, octrees are frequently used, for instance in computational geometry and computer graphics (see e.g. [5]).

An octree is a binary partition of the unit cube. More precisely, an octree is obtained by starting from a single cube, and recursively subdividing cubes into eight smaller cubes of equal size. The structure of an octree can be seen as a tree, where each node represents a cube in the subdivision process. If a cube is further subdivided, the corresponding node in the tree has eight children. The cubes associated to children nodes are also referred to as *sub-cubes* of the cube that corresponds to the parent node. An example of an octree is shown in Figure 1(a).

We say that two cubes in an octree are *face-adjacent* iff they share a two-dimensional face (that is, a face of one of the cubes contains a face of the other cube). Similarly, two cubes are called *edge-adjacent* iff they share a one-dimensional face, and *vertex-adjacent* iff they share a zero-dimensional face. Whenever the term “*adjacent*” appears without further qualification in the following, we refer to face-adjacency.

The purpose of this paper is to analyze the problem of coloring the leaves of octrees (i.e., those cubes that are not further subdivided) using as few colors as possible, such that no two adjacent cubes receive the same color. Figure 1(b) shows a proper coloring

[★] This work was initiated at the 1st Gremo Workshop on Open Problems (GWOP), held at Hof de Planis, Stels, Switzerland, July 8–11, 2003.

^{★★} Supported by the joint Berlin-Zürich graduate program “Combinatorics, Geometry, and Computation”, financed by the German Science Foundation (DFG) and ETH Zürich.

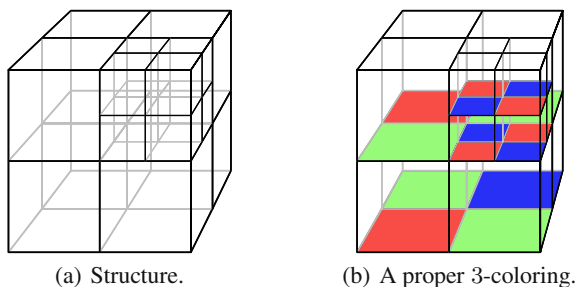


Fig. 1. An octree.

of an octree using three colors. In all figures appearing in this paper we indicate the color of a cube by coloring its bottom face.

In an octree, adjacent cubes may differ in size. To capture this property, we introduce a parameter that measures the unbalance. An octree is said to be k -unbalanced, if the edge lengths of neighboring cubes differ by at most a factor of 2^k . A 0-unbalanced octree is called *balanced*. If there is no condition on the balance of adjacent cubes, we call an octree ∞ -unbalanced, or just unbalanced. As it turns out, the number of colors needed for coloring an octree depends on this parameter.

1.1 Related Work

The problem of coloring octrees has been formulated by Benantar et al. [1]. It is a natural generalization of the corresponding problem for quadrees, which are the two-dimensional counterparts of octrees obtained by recursively subdividing a square into four congruent smaller squares.

For quadrees the coloring problem appears in the context of solving linear elliptic partial differential equations on shared-memory parallel computers [2]. The basic idea is the following. If the squares of a quadtree are colored with respect to vertex-adjacency, then the squares that belong to one particular color class are spatially well separated and can therefore be processed in parallel without conflict. They present a linear-time algorithm for coloring 1-unbalanced quadrees with at most six colors. For unbalanced quadrees an algorithm using at most eight colors is given by Benantar et al. [1].

Recently, Eppstein, Bern and Hutchings [3] proved several results on coloring quadrees with respect to edge-adjacency. They prove that balanced quadrees can be colored with three colors, and that unbalanced quadrees can be colored using at most four colors. Both bounds are tight in the sense that some quadrees require this many colors. They also improved the bound for coloring the squares of unbalanced quadrees with respect to vertex-adjacency from eight to six colors, and constructed a quadtree that requires at least five colors.

1.2 Our Results

In this paper we mainly concentrate on octree coloring problems with respect to face adjacency, and quadtree coloring problems with respect to edge adjacency.

However, recursive subdivisions of cubes are obviously not restricted to dimensions two and three. Since most of our results naturally extend to cubes of arbitrary dimension, we present them in a more general setting. The following definition provides the generalization of octrees to arbitrary dimensional structures, which we call 2^d -trees.

Definition 1. For $d \in \mathbb{N}$, a 2^d -tree is a data structure formed by starting from a single d -dimensional cube and recursively subdividing d -dimensional cubes into 2^d smaller d -dimensional cubes of equal size.

Let k be a non-negative integer. A 2^d -tree is called k -unbalanced iff any two cubes that share a face (that is, a face of one of the cubes contains a face of the other cube) are within a factor of 2^k in edge length.

By coloring a 2^d -tree we mean coloring its *leaves*, that is, those cubes that are not further subdivided. The coloring of a 2^d -tree is called proper iff any two cubes that share a face are colored with different colors. We say that a 2^d -tree is k -colorable iff there exists a proper coloring for its leaves using at most k colors.

The first observation that can be made is that a balanced 2^d -tree is 2-colorable, for every $d \in \mathbb{N}$. Indeed, since all cubes in a balanced 2^d -tree have the same size, it can be represented as a finite set of cells of the axis parallel integer grid subdivision in \mathbb{R}^d . The cells of this subdivision can be colored with two colors, by coloring all cells that have even sum of their coordinates with color one, and all the others with color two. We will refer to this coloring as the *coloring in checkered fashion*.

Our results are presented in Table 1. For each d and k , the corresponding entry c in the table gives the minimal number of colors needed to color a k -unbalanced 2^d -tree. This means that, on one hand, we provide an algorithm for coloring k -unbalanced 2^d -trees with c colors and, on the other hand, we give an example of a k -unbalanced 2^d -tree which cannot be colored with $c - 1$ colors.

The paper of Eppstein, Bern and Hutchings [3] already settles this question for $d = 2, k = 1$, and $d = 2, k = \infty$. These two results are marked bold in the table.

Table 1. Minimal number of colors needed to color a k -unbalanced 2^d -tree.

$d \backslash k$	0	1	2	3	4	5	6	∞
2	2	3	4	4	4	4	4	4
3	2	4		5	5	5	5	5
4	2	4			6	6	6	6
5	2	4	≤ 6			7	7	7
6	2	4	≤ 6				8	8
7	2	4	≤ 6	≤ 8				9

Obviously, the values in the rows of the table are nondecreasing. The same holds for columns, since by using the structure of a k -unbalanced 2^d -tree which is not c -colorable one can easily construct a k -unbalanced 2^{d+1} -tree which is not c -colorable. These properties set upper and lower bounds in the cases in which we could not determine the exact number of colors needed (empty cells).

2 Lower Bounds

Bern, Eppstein and Hutchings [3] gave a neat algorithm to color any 1-unbalanced quadtree using at most three colors. Moreover, the color of each square in the quadtree can be set without knowledge of the subdivision.

For 1-unbalanced octrees, it turns out that three colors are not enough. For the proof of this claim we need the following two lemmata.

Lemma 2. *Let A , B and C be congruent cubes such that both B and C share a face with A , and B and C share an edge. Moreover, let the cube A be further subdivided into eight cubes (see Fig. 2(a)).*

If these cubes are properly colored with three colors, then B and C have to get the same color, and the six sub-cubes of A that are adjacent to B or C have to be colored in checkered fashion with the remaining two colors.

Proof. Suppose that the arrangement of cubes in Figure 2 is properly 3-colored. We denote the two sub-cubes of A that are adjacent to both B and C by A_1 and A_2 . Since

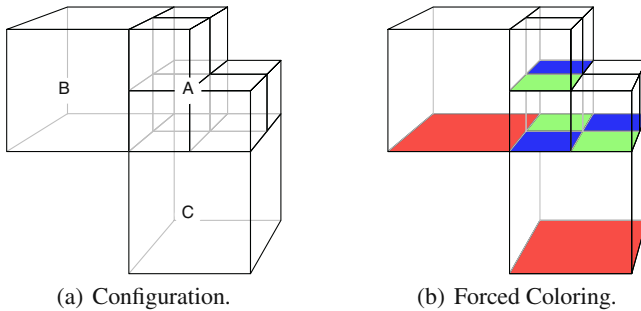


Fig. 2. Illustration for Lemma 2.

B , A_1 and A_2 are pairwise adjacent they all have to be colored with different colors. We can assume without loss of generality that B is colored with color 1, A_1 with color 2, and A_2 with color 3. Since C is adjacent to both A_1 and A_2 , it has to be colored with color 1. Hence, each of the remaining four sub-cubes of A that are adjacent to B or C has to be colored either with color 2, if it is adjacent to A_2 , or with color 3, if it is adjacent to A_1 . That gives exactly the coloring of the six sub-cubes of A by colors 2 and 3 in checkered fashion, as shown in Figure 2(b). \square

Lemma 3. *Let A , B , C and D be congruent cubes, such that all of B , C , and D share a face with A , and that B and C do not share an edge. Moreover, let cube A be further subdivided into eight cubes (see Fig. 3).*

If these cubes are properly colored with three colors, then B , C , and D get the same color, and the eight sub-cubes of A are colored in checkered fashion with the remaining two colors.

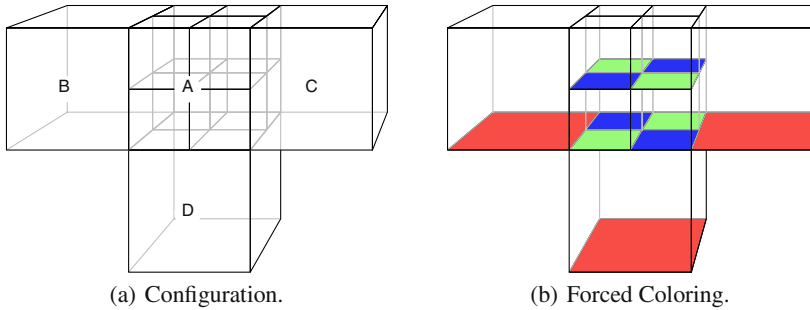


Fig. 3. Illustration for Lemma 3.

Proof. Apply Lemma 2 twice, once to the cubes A , B , and D , and then to the cubes A , C , and D . \square

Note that a collection of cubes can be colored in checkered fashion with two fixed colors in *two* different ways. Using the configurations from the last two lemmata as building blocks, we give a lower bound on the number of colors needed to properly color 1-unbalanced octrees.

Theorem 4. *There exists a 1-unbalanced octree that is not 3-colorable.*

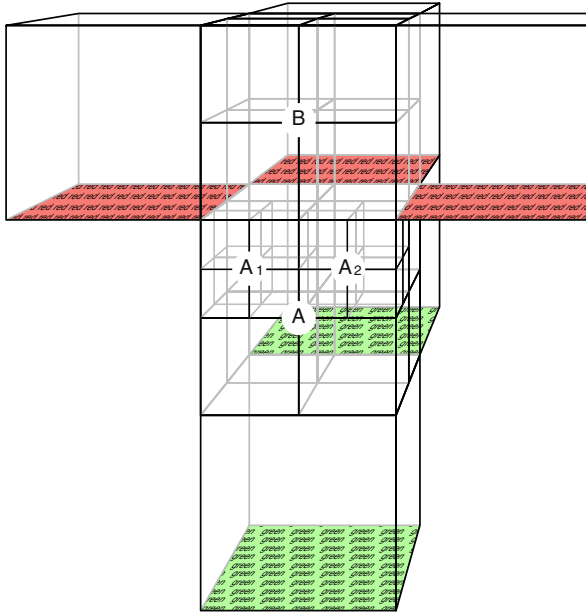
Proof. We show that the 1-unbalanced arrangement of cubes presented in Fig. 4 cannot be colored with three colors. Some cubes of the octree are omitted, since they will not be needed for the analysis. However, they can be easily added without violating the 1-unbalancedness. Note that the cubes that appear in the example are of three different sizes—we will refer to them as big, medium and small cubes. Denote the two big cubes that are subdivided by A and B , and denote the two medium sub-cubes of A that are subdivided into small cubes by A_1 and A_2 , as shown in Fig. 4(a).

Now, suppose there is a proper 3-coloring. First, we look at the two adjacent big cubes at the back of the arrangement (behind A and B). They have to be colored with different colors. We can assume without loss of generality that the top cube is colored with color 1 and the one below is colored with color 2.

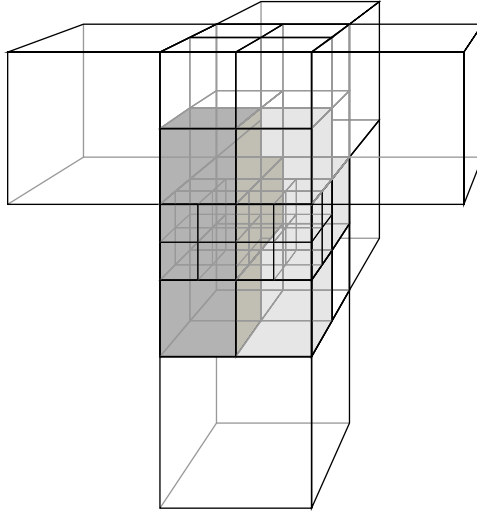
Then, apart from A there are three big cubes adjacent to the subdivided cube B . This arrangement enables us to use Lemma 3. Since we know that the cube behind B has color 1, the other two big cubes should also be colored with color 1. Further, the sub-cubes of B have to be colored in checkered fashion with colors 2 and 3.

Finally, the sub-cubes of A (except A_1 and A_2) together with the two big cubes behind and below A form the configuration from Lemma 2. Since the big cube behind A is colored with color 2, the other one should have color 2 as well. The six medium sub-cubes of A have to be colored in checkered fashion with colors 1 and 3.

It remains to prove that this coloring is not proper. Consider the subdivided cube A_1 and its three neighboring medium cubes that are shaded in Fig. 4(b). By Lemma 3 the medium cubes above and below A_1 both have the same color. Similarly, the medium cubes above and below A_2 both have the same color. But, we already know that the two cubes above A_1 and A_2 are colored in checkered fashion using colors 2 and 3, whereas



(a) The construction



(b) Two configurations to apply Lemma 2.

Fig. 4. A 1-unbalanced octree that is not 3-colorable.

the two cubes below A_1 and A_2 are colored in checkered fashion using colors 1 and 3. Contradiction. \square

Theorem 5. *There exists a d -unbalanced 2^d -tree that cannot be $(d + 1)$ -colored.*

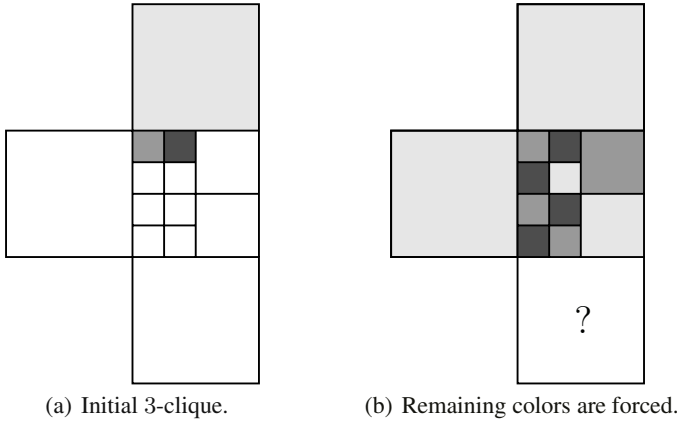


Fig. 5. A 2-unbalanced quadtree that is not 3-colorable.

Proof. We are going to provide examples of a 2-unbalanced quadtree that requires four colors and a 3-unbalanced octree that requires five colors. The latter example can be generalized to prove the statement in higher dimensions.

First, we consider the 2-unbalanced quadtree as shown in Fig. 5. Note that some squares of the depicted quadtree are omitted, as they will not be needed for the analysis. They can be easily added without violating the 2-unbalancedness.

Assume that this quadtree can be 3-colored. The three squares marked in Fig. 5(a) are pairwise adjacent, and, hence, have to be colored differently. If we proceed to color those squares for which the color is uniquely determined, we end up with the coloring depicted in Fig. 5(b). Obviously, this cannot be extended to the last square. Contradiction.

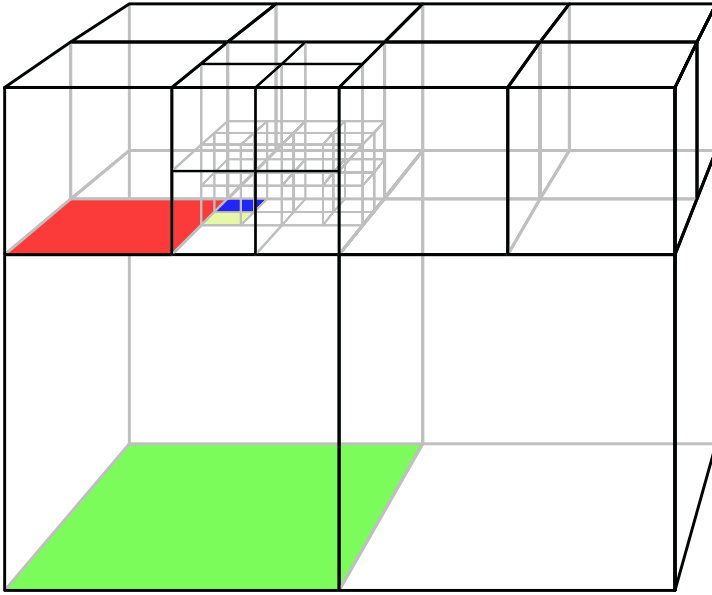
Next, we exhibit a 3-unbalanced octree T that cannot be colored with four colors. The construction is shown in Fig. 6(a).

Assume that this octree can be 4-colored. The four cubes marked in Fig. 6(a) are pairwise adjacent, and, hence, have to be colored differently. If we proceed to color those cubes for which the color is uniquely determined, we end up with the coloring depicted in Fig. 6(b). Obviously, this cannot be extended to the last cube. Contradiction.

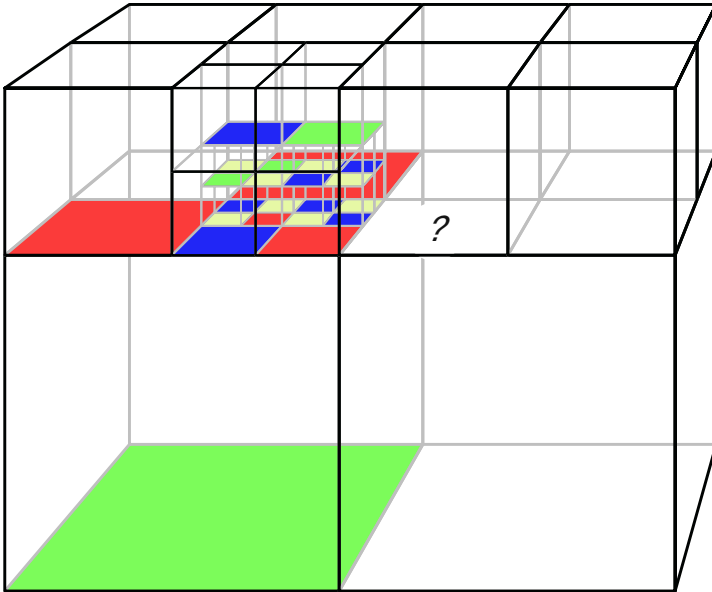
In the following, we use the octree T to construct corresponding examples in higher dimensions. First, we show how to construct a 4-unbalanced 2^4 -tree T' . Start with a 4-dimensional cube C . A copy of T is then placed onto an arbitrary facet of C . We extend (“grow”) each of the 3-dimensional cubes in the copy of T to the 4-dimensional cube, such that it does not overlap with the cube C .

The obtained configuration of 4-dimensional cubes can be easily completed (by adding more cubes) to a 2^4 -tree T' . Since the octree T has only four levels of subdivision, T' is a 4-unbalanced 2^4 -tree.

The cubes in T' obtained from cubes in T are adjacent to one another the same way as in T , and in addition they are all adjacent to the cube C . Since T cannot be colored with four colors, T' cannot be colored with five colors.



(a) Initial 4-clique.



(b) Remaining colors are forced.

Fig. 6. A 3-unbalanced octree that is not 4-colorable.

This construction can be repeated in the same fashion, each time increasing the dimension by 1. This way, for every $d \geq 3$ we obtain a d -unbalanced 2^d -tree that cannot be $(d + 1)$ -colored. \square

3 Algorithms for Coloring 2^d -trees

We proved that at least four colors are needed to color a 1-unbalanced octree. Somewhat surprisingly, four colors are enough even for a 1-unbalanced 2^d -tree of arbitrary dimension.

Theorem 6. *Every k -unbalanced 2^d -tree is $(2k + 2)$ -colorable, for $k \geq 0$, $d \geq 2$.*

Proof. We color the cubes of the k -unbalanced 2^d -tree according to their size—all the cubes with edge length $2^{-\ell}$ are colored in checkered fashion using the colors $(2\ell \bmod 2k + 2)$ and $(2\ell + 1 \bmod 2k + 2)$, for every non-negative integer ℓ .

It remains to be shown that this coloring is proper. Two cubes of different size that are colored with the same color have edge ratio at least 2^{k+1} , and therefore they cannot be adjacent in a k -unbalanced 2^d -tree. If two adjacent cubes have the same size then they are colored with different colors by definition of the coloring in checkered fashion. Hence, the coloring is proper. \square

The upper bound we just obtained is tight in some cases, but not in general. The theorem below provides a better upper bound, if $d < 2k$.

Theorem 7. *Every 2^d -tree is $(d + 2)$ -colorable, for $d \geq 2$.*

Proof. We follow the idea presented in [3]. First, we assign an arbitrary color to the unit cube. Then, we subdivide it recursively assigning colors to the newly obtained cubes—in step t we color all cubes of edge size 2^{-t} . Note that this way cubes are colored even if they are subdivided. Eventually, we obtain a coloring of all leaves of the 2^d -tree.

For each subdivision of a cube of color c into 2^d sub-cubes, we color them as follows. A pairwise non-adjacent half of them (one color class of the checkered 2-coloring) is colored with color c . After that each of the cubes in the other half has d neighbors of color c and at most d additional neighbors of arbitrary color (since at that moment no cube of smaller size has been colored). Altogether, there are at most $d + 1$ different colors in its neighborhood. Hence, one of the $d + 2$ colors can be used to color it. \square

Note that the existence of a coloring for the case $d = 2$ (quadrees) follows from the four-color theorem for planar graphs [4].

4 Open Problems

It remains to determine the exact number of colors needed to color 2-unbalanced octrees (Is it four or five?). We conjecture that four colors are enough.

Another interesting direction is to investigate the octree coloring problems with respect to edge/vertex-adjacency, where to the best of our knowledge no results have been obtained so far.

References

1. M. Benantar, U. Dogrusöz, J. E. Flaherty, and M. S. Krishnamoorthy. Coloring quadrees. Manuscript, 1996.

2. M. Benantar, J. E. Flaherty, and M. S. Krishnamoorthy. Coloring procedures for finite element computation on shared-memory parallel computers. In *Adaptive, Multilevel, and Hierarchical Computation Strategies*, AMD 157, pages 435–449, New York, 1992.
3. M. W. Bern, D. Eppstein, and B. Hutchings. Algorithms for coloring quadrees. *Algorithmica*, 32(1):87–94, 2002.
4. N. Robertson, D. Sanders, P. Seymour, and R. Thomas. The four-colour theorem. *J. Combin. Theory Ser. B*, 70(1):2–44, 1997.
5. H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

Some Open Problems in Decidability of Brick (Labelled Polyomino) Codes

Małgorzata Moczurad and Włodzimierz Moczurad

Institute of Computer Science, Jagiellonian University,
Nawojki 11, 30-072 Kraków, Poland
{mmoczurad,wkm}@ii.uj.edu.pl

Abstract. Bricks are polyominoes with labelled cells. The problem of whether a given set of bricks is a code is undecidable in general. It is open for two-element sets. Here we consider sets consisting of square bricks only. We show that in this setting, the codicity of small sets (two bricks) is decidable, but 15 bricks are enough to make the problem undecidable. Thus the frontier between decidability and undecidability lies somewhere between these two numbers. Additionally we know that the codicity problem is decidable for sets with keys of size n when $n = 1$ and, under obvious constraints, for every n . We prove that it is undecidable in the general case of sets with keys of size n when $n \geq 6$.

1 Introduction

Tiling problems, including polyomino tilings, are widely studied in the literature (e.g. [1, 2]). In this paper we study decidability properties of brick codes which are a natural extension of polyomino codes.

Let A be a finite alphabet of labels. A *brick* is a partial mapping $k : \mathbf{Z}^2 \rightarrow A$, where $\text{dom } k$ is finite and connected. It can be viewed as a polyomino with its cells labelled with the symbols of A . If $|A| = 1$, there is an obvious natural correspondence between bricks and polyominoes. The set of all bricks over A is denoted by A^\boxtimes .

Given a set of bricks $X \subseteq A^\boxtimes$, the set of all bricks tilable with (translated copies of) the elements of X is denoted by X^\boxtimes . Note that we do not allow rotations of bricks. $X \subseteq A^\boxtimes$ is a *brick code*, if every element of X^\boxtimes admits exactly one tiling with the elements of X .

The *effective alphabet* of $X \subseteq A^\boxtimes$ is the set of all symbols that appear on bricks in X , i.e., $\bigcup_{k \in X} k(\text{dom } k)$. If $k \in A^\boxtimes$ is a square brick, then by $\text{len } k$ we denote the edge length of k , i.e., $\sqrt{|\text{dom } k|}$.

Given a rectangular brick $t \in A^\boxtimes$, by $t^{p \times q}$ we denote a brick obtained by stacking together p copies of t vertically, and then q copies of this compound horizontally.

The problem of whether a given set of bricks (even polyominoes) is a code is undecidable in general. This can be proved by reduction from the Wang tilability problem (see [2, 8]). The problem is open for two-element sets. In this paper we

consider sets consisting of square bricks only. We show that in this setting, the codicity of small sets (two bricks) is decidable, but 15 bricks are enough to make the problem undecidable.

Note that the codicity problem is trivially decidable if the squares use just one label, i.e., when they are effectively polyominoes. Only singleton sets are codes then. Thus in the sequel we consider sets of square bricks with an effective alphabet of at least two symbols.

Also note that apart from the single-label case, the decidability of sets of squares does not depend on the size of the effective alphabet, since a larger alphabet can be “simulated” with two symbols at the expense of the size of the squares. When arbitrary shapes are considered, shapes can be used to simulate labels, thus making brick decidability equivalent to polyomino decidability.

In the final section of the paper we investigate the decidability problem for sets with keys, i.e., squares having different top-left “prefixes” of a given size.

2 The Case of Two Square Bricks

We now consider sets consisting of just two bricks, each of them being a square (in the geometrical sense). We show that there exists a simple algorithm to verify whether a given set of this kind is a brick code.

Proposition 1. *Let $X = \{k, l\} \subseteq A^{\boxtimes}$, where $\text{dom } k$ and $\text{dom } l$ are squares, $k \neq l$. Then X is not a brick code iff k and l have a common rectangular tiler, i.e., there exists a rectangle $t \in A^{\boxtimes}$ such that $k, l \in \{t\}^{\boxtimes}$.*

Proof. The “if” part is obvious. Now, if $|\text{dom } k| = |\text{dom } l|$, then k and l have identical shape and differ in their labelling, so X is obviously a code with no common tiler for k and l . Thus we are left with the case, e.g., $|\text{dom } k| < |\text{dom } l|$.

Assume that X is not a brick code. There exists $y \in X^{\boxtimes}$ such that y admits two different tilings with the elements of X .

Take the leftmost cell in the uppermost row of y . If both tilings use the same square $x \in \{k, l\}$ to cover this cell, then remove x from y and repeat this procedure until y' is obtained such that the two tilings place different squares in the leftmost cell in the uppermost row of y' . In other words, we take y to be a minimal brick admitting two tilings over X . Call the tilings T_k and T_l , according to the square being used to cover the cell specified above.

Now this implies that k tiles the top-left corner of l . Moving rightwards along the top edge of l , we observe that when T_k covers the remaining part of l , it cannot place a tile higher than the top edge of l , since we have started in the uppermost row. Thus the next tile to the right of k has to be aligned along the top edge. Since we already know that l contains a copy of k in its top-left corner, we have another copy of k “along the way” (although l may have actually been used). Continuing in this way, we observe that the two tilings will eventually arrive at a common right edge when they reach the lowest common multiple of the edge lengths of k and l .

$$\begin{array}{c} \boxed{k_1 \quad \dots \quad k_n \quad k_1 \quad \dots \quad k_i} \quad k_1 \quad k_2 \quad \dots \quad k_{n-i} \\ \boxed{k_1 \quad \dots \quad k_n} \quad k_1 \quad \dots \quad k_i \quad k_{i+1} \quad k_{i+2} \quad \dots \quad k_n \end{array}$$

Considering the situation to the right of the first copy of l in T_l and denoting the columns of k by k_1, k_2, \dots, k_n we obtain $k_1 = k_{i+1}$, $k_2 = k_{i+2}$, ..., $k_{n-i} = k_n$ where $i = (\text{len } l) \bmod (\text{len } k)$ (cf. figure above). If $i = 0$, then $\text{len } k$ is the width of the common tiler. Otherwise this width is equal to i .

The above argument can be repeated in vertical direction, thus giving a square that can be tiled with k or with l . The size of the square will be the lowest common multiple of the sizes of k and l . \square

Note that the proof becomes trivial if the effective alphabet of X is just one symbol, since X is never a code then with, e.g., the unit square being the common tiler for k and l

Example 1. Consider $X = \{k, l\}$ containing two bricks depicted below. They have a common tiler t , hence X is not a code.

$$\begin{array}{c} \boxed{\begin{array}{ccccc} a & b & a & b & \\ b & b & b & b & \\ a & b & a & b & \\ b & b & b & b & \end{array}} & \boxed{\begin{array}{ccccc} a & b & a & b & a & b \\ b & b & b & b & b & b \\ a & b & a & b & a & b \\ b & b & b & b & b & b \\ a & b & a & b & a & b \\ b & b & b & b & b & b \end{array}} & t = \boxed{\begin{array}{cc} a & b \\ b & b \end{array}}$$

Proposition 2. *If $k, l \in A^{\bowtie}$ have a common rectangular tiler, then they have a common square tiler.*

Proof. Assume that $k, l \in \{t\}^{\bowtie}$, where t is a rectangle of size $p \times q$. Let r be the lowest common multiple of p and q . Both $\text{len } k$ and $\text{len } l$ have to be multiples of r . Hence, $t^{(r/p) \times (r/q)}$ can be taken as the common square tiler. \square

Corollary 1. *Let $X = \{k, l\} \subseteq A^{\bowtie}$, where $\text{dom } k$ and $\text{dom } l$ are squares. It is decidable whether X is a brick code.*

3 The Case of 15 Square Bricks

We show that a Thue system can be reduced to a brick code problem with square bricks. Choosing a small Thue system with an undecidable word problem, we obtain a set of 15 squares, thus proving that the codicity of a set containing 15 square bricks is undecidable.

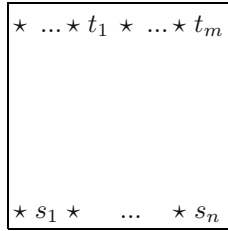
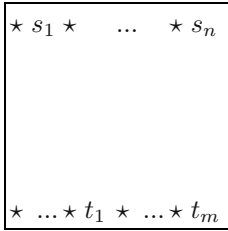
There exists a non-erasing Thue system with an undecidable word problem over a two-letter alphabet with just three relations. This is the smallest example known to us, due to Matiyasevich [5, 6]. We can encode this system, including

two arbitrary input words v and w , in a set X containing 15 square bricks and having the following property: v and w are equivalent (in the Thue sense) iff X is not a brick code. This implies undecidability of the codicity problem for 15 squares.

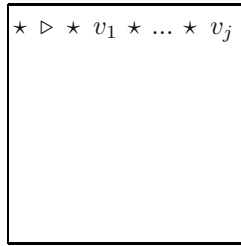
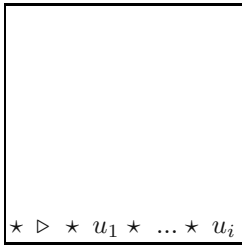
Consider a finite alphabet Σ and a finite subset $S \subseteq \Sigma^* \times \Sigma^*$. The elements of S are called relations. A *Thue system* is the pair (Σ, S) . For $u, v \in \Sigma^*$, we write $u \sim_S v$ if there exists a relation $(s, t) \in S$ or $(t, s) \in S$ such that $u = xsy$ and $v = xty$ for some $x, y \in \Sigma^*$. By \equiv_S we denote the transitive closure of \sim_S . The word problem is: given $u, v \in \Sigma^*$, does $u \equiv_S v$ hold?

For a given Thue system (Σ, S) and two words $u, v \in \Sigma^*$ we construct a set $X_{S,u,v}$ of square bricks over an alphabet $A = \Sigma \cup \{\star, \triangleright\}$ in the way shown below. Cells which are left blank in the diagrams should be labelled with \star .

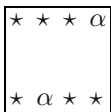
- For each relation $(s_1 \dots s_n, t_1 \dots t_m) \in S$ we draw two bricks. Note that if one of the words is shorter, it is left-padded with \star symbols:



- For the two words $u = u_1 \dots u_i$ and $v = v_1 \dots v_j$ we draw:



- For each symbol $\alpha \in \Sigma \cup \{\triangleright\}$ we draw (“rewriting bricks”):



- Finally, we draw an additional square (a “filler”):

Proposition 3. *Let (Σ, S) be a Thue system and let $u, v \in \Sigma^*$. Let $X_{S,u,v}$ be the set constructed as described above. The following equivalence holds: $u \equiv_S v$ iff $X_{S,u,v}$ is not a brick code.*

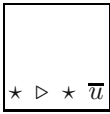
Proof. (\Rightarrow) We first present a sketch of the proof, and then give the details.

A successful derivation in the Thue system corresponds to a series of bricks, stacked vertically in layers: one of the input words, bricks corresponding to the first derivation step, bricks corresponding to the second step, ..., the other input word. Adjacent bricks have the same labels along level boundaries. Rewriting bricks are used to shift non- \star symbols to the left and to rewrite symbols that are not changed at each step.

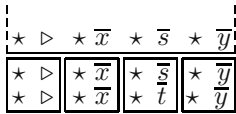
Now note that a tiling described above can also be constructed using the symbol-rewriting bricks and the filler. This implies non-codicity of the set.

To analyze the details of this construction, consider $u \equiv_S v$. Let $\overline{\omega}$ denote the sequence of symbols $\omega_1 \star \omega_2 \star \dots \star \omega_n$, where $\omega = \omega_1 \omega_2 \dots \omega_n \in \Sigma^*$. Using the Thue derivation, we construct a shape as follows:

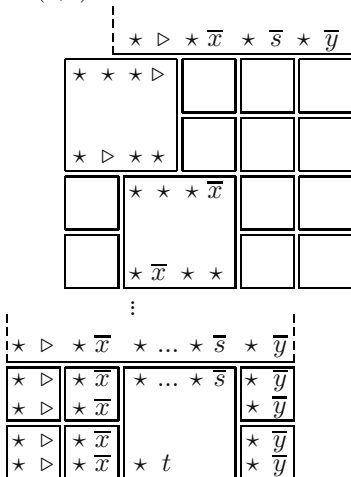
1. Initial lines are the brick corresponding to u :



2. If the derivation includes $u' \sim_S u''$, there exists a relation (s, t) such that $u' = xsy$ and $u'' = xty$. Consider the cases:
 - If $|s| = |t|$, a brick is added as follows:



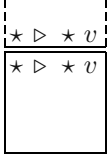
- If $|s| < |t|$, \star symbols have to be added before a brick that corresponds to (s, t) :



- If $|s| > |t|$, a brick is added in a similar way but \star symbols are removed afterwards.

Note that in each case after a brick corresponding to (s, t) is added, the bottom row corresponds to u'' .

3. If the bottom row corresponds to v , the construction is complete:



Another tiling for the shape being constructed can be made with the rewriting bricks and the filler (cf. Example 2).

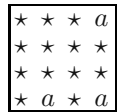
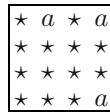
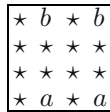
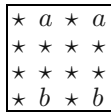
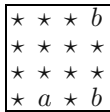
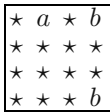
(\Leftarrow) If $u \equiv_S v$ does not hold, there is no word u' such that $u' \sim_S v$, and consequently it is not possible to complete the construction as in case 3. \square

Note that the reduction works even if the Thue system is erasing. A semi-Thue system could also be used with a minor modification (no symmetrical relation bricks). Matiyasevich and S  nizergues give an example of a semi-Thue system over a three-letter alphabet with three relations that has, e.g., an undecidable individual accessibility problem (see [7]). This leads to a similar result.

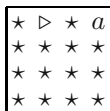
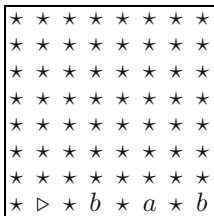
Example 2. The Thue system of Matiyasevich contains a relation of considerable length; thus it is not well-suited as an example. To clarify the idea of the Thue-to-brick reduction, we present a Thue system with short relations. Let $S = \{(ab, b), (aa, bb), (aa, a)\}$. We ask whether $bab \equiv_S a$. The obvious answer is yes, thus the set of bricks X for this Thue system is not a code.

The set X consists of the following bricks:

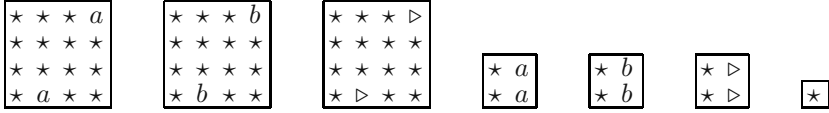
- bricks corresponding to the rules:



- bricks corresponding to the input words:



– rewriting bricks and the filler:



The following derivation corresponds to a tiling shown in Fig. 1. Underlined subwords are the ones being substituted at each step.

$$\underline{bab} \sim_S \underline{bb} \sim_S \underline{aa} \sim_S a$$

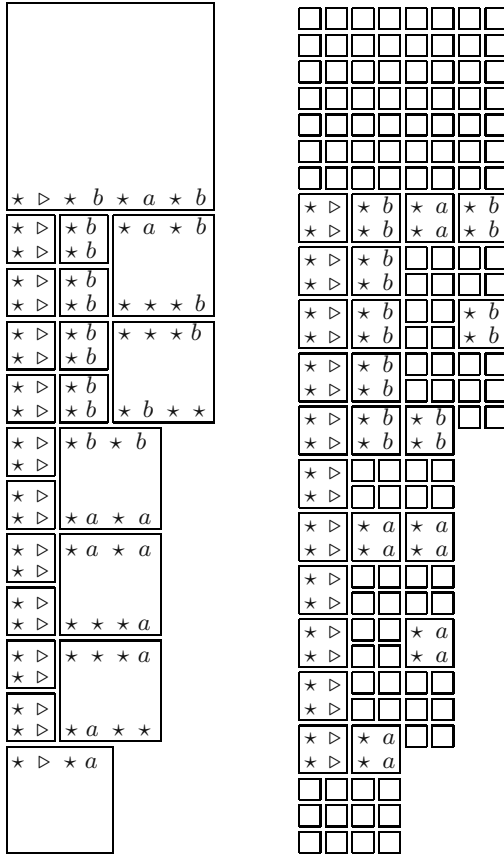


Fig. 1. Tilings corresponding to $\underline{bab} \sim_S \underline{bb} \sim_S \underline{aa} \sim_S a$

Corollary 2. *The codicity problem for sets containing 15 or more square bricks is undecidable.*

Proof. The reduction of a Thue system with three relations over a two-letter alphabet produces six bricks corresponding to the relations, two corresponding to the input words, six rewriting ones, and the filler. \square

4 Decidability of the Codicity Problem for Sets with Keys

Given $k \in A^{\bowtie}$, by $\text{pref}_n(k)$ we denote the $n \times n$ top-left subsquare of k . If $\text{len } k < n$, we define $\text{pref}_n(k) = k$. We say that $X \subseteq A^{\bowtie}$ is a *set with keys of size n* , if $\forall k, l \in X : (\text{pref}_n(k) = \text{pref}_n(l) \Rightarrow k = l)$.

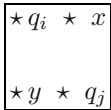
Proposition 4. *Let $X \subseteq A^{\bowtie}$ be a set with keys of size n . If all bricks in X have edge lengths greater or equal to n , then X is a code.*

Proof. Obvious, since each brick has a unique top-left corner. \square

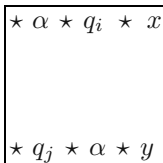
Proposition 5. *Let $n \geq 6$. It is undecidable whether a set with keys of size n is a code.*

Proof. Let $M = (Q, \Sigma, F, \delta, q_0)$ be a Turing machine, where Q is a set of states, Σ is a tape alphabet with a special blank symbol ∇ , F is a set of terminal states, δ is a transition function and q_0 is an initial state. We assume that the tape is infinite to the right, and after the input word there are only blank symbols on the tape. We can encode a Turing machine M in a set $X_{M,w}$ with keys of size 6 over an alphabet $\Sigma \cup Q \cup \{\triangleright, \star\}$ in the following way.

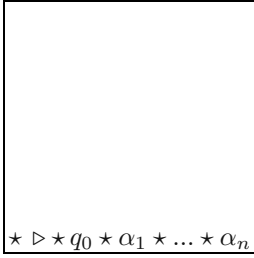
- For each instruction from δ of the form (q_i, x, y, R, q_j) (meaning: if x is read in state q_i , write y , move to the right and change state to q_j) we draw a brick:



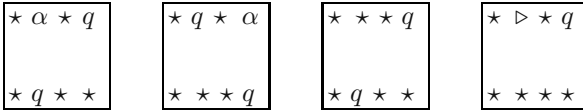
- For each instruction (q_i, x, y, L, q_j) and for each symbol $\alpha \in \Sigma$:



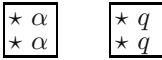
- For the input word $w = \alpha_1 \dots \alpha_n$:



- For each terminal state $q \in F$ and for each symbol $\alpha \in \Sigma$:



- Rewriting bricks for each symbol $\alpha \in \Sigma \cup \{\triangleright\}$ and for each state $q \in Q$:



- Additionally:



The set $X_{M,w}$ is not a code if and only if M stops on the word w . \square

Note that sets where no square is a prefix of another square are obviously codes.

Corollary 3. *The codicity problem for sets with keys of size n is decidable for $n = 1$ and undecidable for $n \geq 6$.*

5 Conclusions

What we know about the decidability of the codicity problem for square bricks amounts to the following:

- If the effective alphabet is trivial, i.e., only one label is used, the problem is trivially decidable.

- If the effective alphabet contains at least two symbols, we have decidability for two-element sets and undecidability for sets with at least 15 elements.
- The problem is trivially decidable for sets with keys of size n with all bricks having edge lengths of at least n .
- The problem is undecidable in the general case of sets with keys of size n , for $n \geq 6$.

We are of course interested in finding exact frontiers between decidability and undecidability (cf. [4]), both with respect to the number of bricks (3...14) and the size of keys (2...5).

References

1. Aigrain, P., Beauquier, D.: Polyomino tilings, cellular automata and codicity. Theoret. Comp. Sci. **147** (1995) 165–180
2. Beauquier, D., Nivat, M.: A codicity undecidable problem in the plane. Theoret. Comp. Sci. **303** (2003) 417–430
3. Karhumäki, J.: Some open problems in combinatorics of words and related areas. TUCS Technical Report **359** (2000)
4. Margenstern, M.: Frontiers between decidability and undecidability: a survey. Theoret. Comp. Sci. **231** (2000) 217–251
5. Matiyasevich, Yu.: Simple examples of undecidable associative calculi. Soviet Math. Dokladi **8** (1967) 555–557
6. Matiyasevich, Yu.: Word problem for Thue systems with a few relations. Lecture Notes in Computer Science, Vol. 909. Springer-Verlag (1995) 39–53
7. Matiyasevich, Yu., Sénizergues, G.: Decision problems for semi-Thue systems with a few rules. Proceedings LICS'96 (1996) 523–531
8. Moczurad, W.: Algebraic and algorithmic properties of brick codes. Doctoral thesis, Jagiellonian University (1999)
9. Moczurad, W.: Brick codes: families, properties, relations. Intern. J. Comp. Math. **74** (2000) 133–150

Q -Ary Ulam-Rényi Game with Weighted Constrained Lies^{*}

Ferdinando Cicalese¹, Christian Deppe², and Daniele Mundici³

¹ Department of Computer Science and Applications, University of Salerno,
via S. Allende, 84081 Baronissi (SA), Italy
cicalese@dia.unisa.it

² Faculty of Mathematics, University of Bielefeld,
Postfach 100131, D-33501 Bielefeld, Germany
cdeppe@mathematik.uni-bielefeld.de

³ Department of Mathematics, University of Florence,
Viale Morgagni 67/A, 50134 Florence, Italy
mundici@math.unifi.it

Abstract. The Ulam-Rényi game is a classical model for the problem of determining the minimum number of queries to find an unknown number in a finite set when up to a finite number of the answers may be erroneous/mendacious. In the variant considered in this paper, questions with q many possible answers are allowed, with q fixed and known beforehand; further, lies are constrained by a weighted bipartite graph (the “channel”). We provide a tight asymptotic estimate for the number of questions needed to solve the problem. Our results are constructive, and the appropriate searching strategies are actually provided. As an extra bonus, all our strategies use the minimum amount of adaptiveness: they ask a first batch of nonadaptive questions, and then, only depending on the answers to these questions, they ask a second nonadaptive batch.

1 Introduction

In the q -ary Ulam-Rényi game two players, called Paul and Carole, first agree on fixing an integer $M \geq 1$ and a search space $U = \{0, \dots, M-1\}$. Then Carole chooses a number $x_* \in U$, the secret number, and Paul must find out x_* by asking the minimum number of q -ary questions. By a q -ary *question* we understand a list T_0, \dots, T_{q-1} of pairwise disjoint subsets forming a partition of the set U . The parameter q is fixed beforehand. When presented such a list, Carole will answer by pointing out the set T_k , supposedly containing the secret number x_* . It is however agreed that Carole may give up to e mendacious (erroneous) answers. The integer $e \geq 0$ is fixed and known to both players. Intuitively, any q -ary question asks “Which set among T_0, T_1, \dots, T_{q-1} does the secret number x_* belong to?” and the answer is just an index $k \in \mathcal{Q} = \{0, 1, \dots, q-1\}$, meaning that x_* belongs to T_k .

We generalize the q -ary game in the following way. Before the game starts, Carole and Paul fix a function $\Gamma : \mathcal{Q} \times \mathcal{Q} \rightarrow \mathbb{N}_0 = \{0, 1, 2, \dots\}$ such that

^{*} This work was partially supported by INTAS 00-738

$\Gamma(i, i) = 0$ for each i , and $\Gamma(i, j) > 0$ for each $i \neq j$. The function Γ is meant to weigh Carole's answers. More precisely, whenever Carole answers j to a question whose sincere answer is i , we say that Carole's answer has *individual weight* $\Gamma(i, j)$. Note that every sincere answer has weight 0. The parameter e is meant to bound Carole's lies via the following new rule:

Carole is allowed to give false answers, *with total weight (up to) e* .

Thus, if k is the honest answer to Paul's first question $\mathbf{T} = \{T_0, T_1, \dots, T_{q-1}\}$, (in the sense that $x_* \in T_k$) then Carole can choose her answer in the set $\{j : e - \Gamma(k, j) \geq 0\}$. If Carole answers i , with i in this set, then her available weight becomes $e - \Gamma(k, i)$. And the *individual weight* of her answer is $\Gamma(k, i)$. By induction, if k' is the correct answer to Paul's current question \mathbf{T}' , and e' is Carole's currently available weight, then she can choose her answer in the set $\{j : e' - \Gamma(k', j) \geq 0\}$. If Carole answers j' , then her available weight reduces to $e' - \Gamma(k', j')$, and the individual weight of this answer is $\Gamma(k', j')$.

We think of Γ as the weighted noise-transition pattern on a channel carrying Carole's answers. The q -ary Ulam-Rényi game with this sort of restriction on the patterns of Carole's lies will be called the *game over the channel Γ* . Each pair (i, j) with $\Gamma(i, j) > e$ stands for an impossible lie: indeed, if Carole answered j to a question whose sincere answer is i , already the individual weight of this answer would exceed the total weight e .

The classical q -ary Ulam-Rényi game is a *Ulam-Rényi game over a channel Γ* , such that $\Gamma(i, j) = 1$ whenever $i \neq j$ (see [2]). Let

$$w_{\min}^{\Gamma} = \min\{\Gamma(i, j) : i \neq j\}, \quad (1)$$

$$E_{\min}^{\Gamma}(k) = \{(j, k) : \Gamma(j, k) = w_{\min}^{\Gamma}\}, \quad (2)$$

$$E_{\min}^{\Gamma} = \bigcup_{k=0}^{q-1} E_{\min}^{\Gamma}(k). \quad (3)$$

We call Γ a *d-rightregular channel*, if $|E_{\min}^{\Gamma}(k)| = d$ for all k . Trivially, in this case $|E_{\min}^{\Gamma}(k)| = |E_{\min}^{\Gamma}|/q$ for each $k = 0, \dots, q-1$. To ease notation we shall write $w_{\min}, E_{\min}(k), E_{\min}$ for $w_{\min}^{\Gamma}, E_{\min}^{\Gamma}(k), E_{\min}^{\Gamma}$, respectively, whenever the channel Γ is clear from the context.

For any choice of the parameters q, e, M, d , and for any d -rightregular channel Γ , let $N_{\Gamma}^{[d, q]}(M, e)$ be the minimum number of questions that Paul must ask in order to infallibly guess a number $x_* \in \{0, 1, \dots, M-1\}$, in the q -ary Ulam-Rényi game with lies, over the channel Γ , with total weight e . In this paper we deal with the dual problem of determining the largest possible $M = M(q, e, \Gamma, n)$ such that for all possible choices of the parameters q, e, n, d , and for any d -rightregular channel Γ , there exists a strategy of size n for Paul to infallibly guess a number $x_* \in \{0, 1, \dots, M-1\}$ in the q -ary Ulam-Rényi game with lies over the channel Γ with total weight e . We consider the case $M = q^m$ ($m = 0, 1, \dots$) and prove that the exponent $m = m(q, e, \Gamma, n)$ (hence, the quantity M) only depend on Γ via the parameter d . Further, all our strategies can be implemented by procedures which use adaptiveness only once.

We generalize the results in [3], where optimal and quasi-optimal strategies (i.e., strategies whose length differs by at most one from the information theoretic lower bound) are introduced for all sufficiently large M and for d -regular channels Γ , satisfying $\Gamma(i, j) \in \{0, 1, e + 1\}$.

The Ulam-Rényi game was introduced independently in [11], [12] (also see [1]). There exist many generalizations of this game, see the surveys [7], [10], [8]. Two-batch strategies were introduced in [4] [5] (see also [6]). In [9] the Ulam-Rényi game over an arbitrary channel Γ was studied under the hypothesis that $\Gamma(i, j) \in \{1, e + 1\}$ for all $i \neq j$.

2 Ulam-Rényi Game on a d -Rightregular Channel

Suppose Paul and Carole first fix integers $q \geq 2$, $0 \leq d < q$, $M \geq 1$ and $e \geq 0$, together with a d -rightregular channel Γ . The search space is identified with the set $U = \{0, 1, \dots, M - 1\}$. Let $\mathcal{Q} = \{0, 1, \dots, q - 1\}$. Carole chooses a number $x_* \in U$ and Paul must guess it by asking as few as possible q -ary questions. Carole is allowed to choose wrong answers with total weight e on the channel Γ . We say that the current weight available to Carole is $e - w$ if the sum of the individual weights of her previous answers equals w .

At any stage of the game, when questions $\mathbf{T}_1, \dots, \mathbf{T}_t$ have been asked and answers $b^t = b_1, \dots, b_t$ have been received (with $b_i \in \mathcal{Q}$) Paul's *state* of knowledge is represented by an $(e + 1)$ -tuple $\sigma = (A_0, A_1, A_2, \dots, A_e)$ of pairwise disjoint subsets of U , where for each $i = 0, 1, 2, \dots, e$ A_i is the set of elements of U which could possibly coincide with x_* supposing that the sum of the individual weights of Carole's answers b_1, \dots, b_t equals i . In particular, the *initial* state σ_0 is given by $(U, \emptyset, \emptyset, \dots, \emptyset)$.

Fix $j = 0, \dots, t - 1$ and write $k = b_{j+1}$. Assume Paul is in state $\sigma_j = (B_0, \dots, B_e)$. Then Paul's new state $\sigma_{j+1} = \sigma_j^k = (C_0, \dots, C_e)$ resulting from Carole's answer k to question \mathbf{T}_{j+1} is given by

$$C_i = \bigcup_{\{j \in \mathcal{Q} : \Gamma(j, k) \leq i\}} (B_{i - \Gamma(j, k)} \cap T_j).$$

By induction, Carole's answers b_1, \dots, b_t determine a sequence of states

$$\sigma_0 = \sigma, \quad \sigma_1 = \sigma_0^{b_1}, \quad \sigma_2 = \sigma_1^{b_2}, \quad \dots, \quad \sigma_t = \sigma_{t-1}^{b_t} = \sigma^{b^t}.$$

A state $(A_0, A_1, A_2, \dots, A_e)$ is *final* iff the set $A_0 \cup A_1 \cup A_2 \cup \dots \cup A_e$ has at most one element. By a *strategy* \mathcal{S} with n questions we mean the q -ary tree of depth n , where each node ν is mapped into a question \mathbf{T}_ν , and the q edges $\eta_0, \eta_1, \dots, \eta_{q-1}$ generated by ν are, respectively from left to right, labeled with $0, 1, \dots, q - 1$, which represent Carole's possible answers to \mathbf{T}_ν . Let $\boldsymbol{\eta} = \eta_1, \dots, \eta_n$ be a path in \mathcal{S} , from the root to a leaf, with respective labels b_1, \dots, b_n , generating nodes ν_1, \dots, ν_n and associated questions $\mathbf{T}_{\nu_1}, \dots, \mathbf{T}_{\nu_n}$. We say that strategy \mathcal{S} is *winning* for σ iff for every path $\boldsymbol{\eta}$ the state $\sigma^{\boldsymbol{\eta}}$ is final. A strategy is said to be *nonadaptive* if all nodes at the same depth of the tree are mapped into the same question.

3 An Upper Bound for the Size of the Search Space

Using a result of [9] in this section we shall give an upper bound on the largest integer M such that Paul has a strategy of size n to infallibly guess Carole's secret number in the Ulam-Rényi game over the channel Γ with total weight e , over a search space of cardinality M . Our result holds for arbitrary Γ and for all sufficiently large n .

Theorem 1. [Dumitriu, Spencer]

Fix integers $q \geq 2$ and $e \geq 0$ and let $\mathcal{Q} = \{0, 1, \dots, q-1\}$. Fix $\Gamma : \mathcal{Q} \times \mathcal{Q} \rightarrow \{0, 1, e+1\}$ such that $\Gamma(i, j) = 0$, iff $i = j$, for each $i, j \in \mathcal{Q}$. Let $E = \{(i, j) \in \mathcal{Q} \times \mathcal{Q} : \Gamma(i, j) > 0\}$. It follows that, for all $\epsilon > 0$ there exists an integer $n_0 > 0$ such that for all $n \geq n_0$, if Paul has a strategy of size n to infallibly guess a number $x_* \in \{0, 1, \dots, M-1\}$ in the q -ary Ulam-Rényi game with lies over the channel Γ , with total weight e , then

$$M \leq \left(\left(\frac{q}{|E|} \right)^e + \epsilon \right) \frac{q^n}{\binom{n}{e}}.$$

As a consequence, recalling the definition of w_{\min}^Γ , $E_{\min}^\Gamma(k)$ and E_{\min}^Γ in (1)-(3) we have:

Theorem 2. Fix integers $q \geq 2$ and $e \geq 0$ and let $\mathcal{Q} = \{0, 1, \dots, q-1\}$. Fix a function $\Gamma : \mathcal{Q} \times \mathcal{Q} \rightarrow \mathbb{N}_0$ such that $\Gamma(i, i) = 0$ for each i and $\Gamma(i, j) > 0$ for $i \neq j$. Then for all $\epsilon > 0$ there exists an integer n_0 , such that for all integers $n > n_0$, if Paul has a strategy of size n to infallibly guess a number $x_* \in \{0, 1, \dots, M-1\}$, in the q -ary Ulam-Rényi game with lies, over the channel Γ , with total weight e , then

$$M \leq \left(\left(\frac{q}{|E_{\min}^\Gamma|} \right)^{\left\lfloor \frac{e}{w_{\min}^\Gamma} \right\rfloor} + \epsilon \right) \frac{q^n}{\binom{n}{\left\lfloor \frac{e}{w_{\min}^\Gamma} \right\rfloor}}.$$

In particular, if Γ is a d -rightregular channel we have

$$M \leq \left(d^{-\left\lfloor \frac{e}{w_{\min}^\Gamma} \right\rfloor} + \epsilon \right) \frac{q^n}{\binom{n}{\left\lfloor \frac{e}{w_{\min}^\Gamma} \right\rfloor}}.$$

Proof. Let $w = w_{\min}^\Gamma$. Let $\Gamma' : \mathcal{Q} \times \mathcal{Q} \rightarrow \{0, w, e+1\}$ be defined by

$$\Gamma'(i, j) = \begin{cases} \Gamma(i, j), & \text{whenever } \Gamma(i, j) \in \{0, w\} \\ e+1, & \text{otherwise.} \end{cases}$$

For any fixed M, e, n , if Paul has no winning strategy with n questions in a game over the channel Γ' , then a fortiori he has no winning strategy for the game over the channel Γ . In fact, in the channel Γ Carole can choose her lies more freely than in Γ' , whence Paul's searching game becomes more difficult. Moreover, in the game over Γ' each one of Carole's lies weighs exactly w . Since the total

weight of her lies cannot exceed e , the maximum number of mendacious answers is at most $\lfloor e/w \rfloor$. Let $\Gamma'' : \mathcal{Q} \times \mathcal{Q} \rightarrow \{0, 1, e+1\}$ be defined by

$$\Gamma''(i, j) = \begin{cases} 1 & \text{whenever } \Gamma'(i, j) = w \\ \Gamma'(i, j) & \text{otherwise.} \end{cases}$$

Trivially, the game over Γ' with total weight e is equivalent to the game over Γ'' with total weight $\lfloor e/w \rfloor$. Thus, for all $n = 1, 2, \dots$, we have

$$M(q, e, \Gamma, n) \leq M(q, e, \Gamma', n) = M(q, \lfloor e/w \rfloor, \Gamma'', n).$$

Then from Theorem 1 we immediately get the desired conclusion.

4 Optimal Strategies with Minimum Adaptiveness for $M = q^m$

In this section we prove the existence of a two-batch-strategy which matches the bound given by Theorem 2. The whole section will be devoted to the proof of the following theorem.

Theorem 3. *Fix integers $e \geq 0$ and $q \geq 2$ and let $\mathcal{Q} = \{0, 1, \dots, q-1\}$. Fix an integer $d \in \mathcal{Q}$ and a function $\Gamma : \mathcal{Q} \times \mathcal{Q} \rightarrow \mathbb{N}$ satisfying $\Gamma(i, i) = 0$ and $|E_{\min}^{\Gamma}(i)| = d$, for each $i \in \mathcal{Q}$. Then for all $\epsilon > 0$ there exists an integer n_0 , such that for all integers $n > n_0$, if*

$$M = q^m \leq \left(d - \left\lfloor \frac{e}{w_{\min}^{\Gamma}} \right\rfloor + \epsilon \right) \frac{q^n}{\left(\left\lfloor \frac{e}{w_{\min}^{\Gamma}} \right\rfloor \right)}, \quad (4)$$

then Paul has a strategy of size n to infallibly guess a number $x_ \in \{0, 1, \dots, M-1\}$, in the q -ary Ulam-Rényi game with lies, over the channel Γ ¹, with total weight e . Moreover, this strategy involves minimum adaptiveness: Paul asks a first batch of nonadaptive questions and then, only depending on Carole's answers to these questions, a second and final batch of nonadaptive questions.*

4.1 The First Batch of Questions

To ease notation we shall write w for w_{\min}^{Γ} . Let the function $\Gamma' : \mathcal{Q} \times \mathcal{Q} \rightarrow \{0, w, w+1\}$ be defined by

$$\Gamma'(i, j) = \begin{cases} \Gamma(i, j) & \Gamma(i, j) \in \{0, w\} \\ w+1 & \text{otherwise.} \end{cases}$$

Recalling the definition of $m(q, e, \Gamma, n)$ we can write

$$m(q, e, \Gamma, n) \geq m(q, e, \Gamma', n).$$

As a matter of fact, in the Ulam-Rényi game over the channel Γ' , with total weight e , each lie has a weight that cannot exceed the weight of a corresponding lie in the Ulam-Rényi game over the channel Γ .

¹ Notice that, in fact, Γ defines a d -right regular channel

Trivially, any winning strategy for the game over channel Γ' with total weight e is also a winning strategy for the game over channel Γ with total weight e .

Henceforth we shall concentrate our attention to the game over the channel Γ' , with total weight e . We shall prove that under the hypothesis (4), there is a two-batch strategy for Paul to infallibly guess a number in the set $\{0, 1, \dots, q^m - 1\}$ in the Ulam-Rényi game over the (d -right regular) channel Γ' with total weight e .

The first batch of questions is defined as follows: For each $j = 1, 2, \dots, m$, let $\mathcal{D}_j = (D_{j0}, D_{j1}, \dots, D_{jq-1})$ denote the question

“What is the j th (q -ary) digit of x_* ?”

More precisely, a number $y \in U$ belongs to D_{ji} iff the j th digit y_j of its q -ary expansion $\mathbf{y} = y_1 \cdots y_m$ is equal to i . Let $b_j \in \{0, 1, \dots, q-1\}$ be the answer to question \mathcal{D}_j . Let $\mathbf{b} = b_1 \cdots b_m$. Starting from the initial state $\sigma = (U, \emptyset, \dots, \emptyset)$, Paul's state resulting from Carole's answers $b_1 \cdots b_m$ is the $(e+1)$ -tuple $\sigma^{\mathbf{b}} = (A_0, A_1, \dots, A_e)$, where for all $i = 0, 1, \dots, e$, $A_i = \{y \in U \mid d_{\Gamma'}(\mathbf{y}, \mathbf{b}) = i\}$ and $d_{\Gamma'}(\mathbf{y}, \mathbf{b}) = \sum_{i=1}^m \Gamma'(y_i, b_i)$. Therefore, A_i is the set of all q -ary m -tuples \mathbf{y} having the following property:

- There exists an integer $j \in \{0, 1, \dots, \lfloor i/w \rfloor\}$ such that $(i - jw)/(w+1)$ is also an integer, and there exist two disjoint subsets of $\{1, 2, \dots, m\}$, say $W = \{k_1, k_2, \dots, k_j\}$ and $W' = \{\ell_1, \ell_2, \dots, \ell_{(i-jw)/(w+1)}\}$, such that

$$\Gamma'(y_u, b_u) = \begin{cases} w & u \in W \\ w+1 & u \in W' \\ 0 & \text{otherwise.} \end{cases}$$

Thus,

$$|A_i| = \sum_{j=0}^{\lfloor \frac{i}{w} \rfloor} \binom{m}{j} g(m-j, \frac{i-jw}{w+1})(q-1-d)^{\frac{i-jw}{w+1}} d^j, \quad (5)$$

where

$$g(x, y) = \begin{cases} \binom{x}{y} & \text{if } x, y \in \mathbb{N}_0 \\ 0 & \text{otherwise.} \end{cases}$$

We conclude that there exists a constant $0 < \gamma < 1$ such that, for all sufficiently large m , and for each $i = 0, 1, \dots, e$,

$$|A_i| \leq \binom{m}{\lfloor i/w \rfloor} q^{\lfloor i/w \rfloor} (1 - \gamma). \quad (6)$$

4.2 Intermezzo: Error Correcting Codes à la Gilbert

Lemma 1. *Let the parameters e, q, d, Γ, m, M and n be as in Theorem 3. Let $w = w_{\min}^{\Gamma}$ and for each $i = 0, 1, \dots, e$, let A_i be a set satisfying (5)-(6). Then we have the inequality*

$$q^{n-m} \geq \left(\sum_{j=0}^{e-w} |A_j| \right) \left(\sum_{i=0}^{2\lfloor e/w \rfloor} \binom{n-m}{i} q^i \right) + \sum_{j=e-w+1}^e |A_j|. \quad (7)$$

Proof. By (4) there exists a constant $0 < \beta < 1$ such that, for all sufficiently large n , and for all m satisfying (4), we have $n \geq m + (1 - \beta) \lfloor \frac{e}{w} \rfloor \log_q n$. It follows that

$$\binom{n}{\lfloor \frac{e}{w} \rfloor} \left(d^{-\lfloor \frac{e}{w} \rfloor} + \epsilon \right)^{-1} > \binom{m}{\lfloor \frac{e}{w} \rfloor} d^{\lfloor \frac{e}{w} \rfloor}. \quad (8)$$

Claim 1. The following inequality holds:

$$m \leq \frac{(\lfloor \frac{e}{w} \rfloor! q^{n-m})^{\frac{1}{\lfloor \frac{e}{w} \rfloor}}}{d} + \left\lfloor \frac{e}{w} \right\rfloor. \quad (9)$$

For otherwise (absurdum hypothesis), using (8), we would have

$$\begin{aligned} \binom{n}{\lfloor \frac{e}{w} \rfloor} \left(d^{-\lfloor \frac{e}{w} \rfloor} + \epsilon \right)^{-1} &> \binom{m}{\lfloor \frac{e}{w} \rfloor} d^{\lfloor \frac{e}{w} \rfloor} \\ &\geq \frac{(m - \lfloor \frac{e}{w} \rfloor)^{\lfloor \frac{e}{w} \rfloor}}{\lfloor \frac{e}{w} \rfloor!} d^{\lfloor \frac{e}{w} \rfloor} \\ &\geq \frac{\left(\frac{(\lfloor \frac{e}{w} \rfloor! q^{n-m})^{\frac{1}{\lfloor \frac{e}{w} \rfloor}}}{d} \right)^{\lfloor \frac{e}{w} \rfloor}}{\lfloor \frac{e}{w} \rfloor!} d^{\lfloor \frac{e}{w} \rfloor} \\ &= q^{n-m} \end{aligned}$$

contradicting (4). Our first claim is settled.

Claim 2. There exists a constant $0 < \gamma' < 1$, such that

$$\sum_{j=e-w+1}^e |A_j| \leq (1 - \gamma') q^{n-m}. \quad (10)$$

Indeed, Claim 1 together with (6) are to the effect that

$$\sum_{j=e-w+1}^e |A_j| \leq \sum_{j=e-w+1}^e \binom{m}{\lfloor i/w \rfloor} q^{\lfloor i/w \rfloor} (1 - \gamma) \leq w(1 - \gamma) m^{\lfloor \frac{e}{w} \rfloor} \leq (1 - \gamma') q^{n-m},$$

for some $0 < \gamma' < 1$. This settles our second claim.

To conclude the proof, recalling (6), there exist polynomials \mathbf{p} and \mathbf{p}_1 , such that

$$\sum_{j=e-w+1}^e |A_j| + \left(\sum_{j=0}^{e-w} |A_j| \right) \left(\sum_{i=0}^{2\lfloor e/w \rfloor} \binom{n-m}{i} q^i \right)$$

$$< (e - w + 1) |A_{e-w}| \left(\sum_{i=0}^{2\lfloor e/w \rfloor} \binom{n-m}{i} q^i \right) + (1 - \gamma') q^{n-m} \quad (11)$$

$$\leq \binom{m}{\lfloor \frac{e}{w} \rfloor - 1} \mathbf{p}(n-m) + (1 - \gamma') q^{n-m} \quad (12)$$

$$\leq \frac{m^{\lfloor \frac{e}{w} \rfloor - 1}}{(\lfloor \frac{e}{w} \rfloor - 1)!} \mathbf{p}(n-m) + (1 - \gamma') q^{n-m} \quad (13)$$

$$\leq \frac{\left(\frac{(\lfloor \frac{e}{w} \rfloor! q^{n-m})^{\lfloor \frac{1}{d} \rfloor}}{d} + \lfloor \frac{e}{w} \rfloor \right)^{\lfloor \frac{e}{w} \rfloor - 1}}{(\lfloor \frac{e}{w} \rfloor - 1)!} \mathbf{p}(n-m) + (1 - \gamma') q^{n-m} \quad (14)$$

$$\leq \mathbf{p}_1(n-m) q^{(n-m) \frac{\lfloor \frac{e}{w} \rfloor - 1}{\lfloor \frac{e}{w} \rfloor}} + (1 - \gamma') q^{n-m} \quad (15)$$

$$< \gamma' q^{n-m} + (1 - \gamma') q^{n-m}. \quad (16)$$

Here,

- (11) follows from the monotonicity of the sizes of the sets A_i , together with Claim 2,
- (12) follows from (6), upon noting that there exists a polynomial in $(n-m)$ bounding $(e-w+1)(1-\gamma)q^{\lfloor \frac{e}{w} \rfloor - 1} \left(\sum_{i=0}^{2\lfloor e/w \rfloor} \binom{n-m}{i} q^i \right)$.
- (13) trivially follows from the properties of the binomial coefficient,
- (14) follows from Claim 1,
- (15) follows because $\frac{\mathbf{p}(n-m) \frac{\lfloor \frac{e}{w} \rfloor! q^{\lfloor \frac{e}{w} \rfloor - 1}}{d^{\lfloor \frac{e}{w} \rfloor - 1}}}{d^{\lfloor \frac{e}{w} \rfloor - 1}}$ is bounded by a polynomial in $n-m$,
- (16) follows choosing n_0 such that for all $n > n_0$, $n > m + \lfloor \frac{e}{w} \rfloor \log_q \frac{p_1(n-m)}{\gamma'}$

The proof is complete.

Lemma 2. *Let the parameters $e, q, d, \Gamma, m, M, n, w_{\min}$ be as in Theorem 3. Let $w = w_{\min}$ and A_i satisfy (5)-(6) for each $i = 0, 1, \dots, e$. Then there exist disjoint sets, $\mathcal{C}_1, \mathcal{C}_2$, of q -ary tuples of length $n-m$ such that*

- (i) $|\mathcal{C}_1| \geq \sum_{i=0}^{e-w} |A_i|$,
- (ii) for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{C}_1$, $d_H(\mathbf{x}_1, \mathbf{x}_2) \geq 2\lfloor \frac{e}{w} \rfloor + 1$,
- (iii) $|\mathcal{C}_2| \geq \sum_{i=e-w+1}^e |A_i|$,
- (iv) for all $\mathbf{x}_1 \in \mathcal{C}_1, \mathbf{x}_2 \in \mathcal{C}_2$, $d_H(\mathbf{x}_1, \mathbf{x}_2) \geq \lfloor \frac{e}{w} \rfloor + 1$,

where $d_H(\cdot, \cdot)$ denotes Hamming distance between q -ary vectors.

Proof. The sets \mathcal{C}_1 and \mathcal{C}_2 will be constructed via the following greedy algorithm. Let $\mathcal{R} = \{0, 1, \dots, q-1\}^{n-m}$. We first build the set $\mathcal{C}_1 \subseteq \mathcal{R}$ by the following procedure:

1. Pick an arbitrary vector $\mathbf{x} \in \mathcal{R}$ and include it in \mathcal{C}_1 .
2. Delete from \mathcal{R} all vectors \mathbf{y} such that $d_H(\mathbf{x}, \mathbf{y}) \leq 2\lfloor \frac{e}{w} \rfloor$.
3. If $|\mathcal{C}_1| \leq \sum_{i=0}^{e-w} |A_i|$ go back to 1.

Each time step 2. is performed, at most $\sum_{j=0}^{2\lfloor e/w \rfloor} \binom{n-m}{j} q^j$ vectors are deleted from \mathcal{R} . Lemma 1 guarantees that, as long as $|\mathcal{C}'_1| \leq \sum_{i=0}^{e-w} |A_i|$, one can add new elements to \mathcal{C}'_1 : indeed, there are more elements in the set $\{0, 1, \dots, q-1\}^{n-m}$ than in the union of \mathcal{C}'_1 and the set of deleted vectors. Once the set \mathcal{C}_1 has been constructed with its $\sum_{i=0}^{e-w} |A_i|$ vectors, by Lemma 1 in $\mathcal{R} \setminus \mathcal{C}_1$ there still exist $\sum_{i=e-w+1}^e |A_i|$ many vectors which have not been discarded during the construction of \mathcal{C}_1 . These vectors will constitute the set \mathcal{C}_2 . By direct inspection, \mathcal{C}_1 and \mathcal{C}_2 satisfy (ii) and (iv). The proof is complete.

4.3 The Second Batch of Questions

To conclude the proof of Theorem 3, we will show that starting from the state σ^b , a second batch of $n-m$ nonadaptive questions is sufficient to guess the secret number x_* in the Ulam-Rényi game over the channel Γ'' with total weight e , where

$$\Gamma''(i, j) = \begin{cases} w & \text{if } i \neq j \\ 0 & \text{otherwise.} \end{cases}$$

The replacement of channel Γ' by channel Γ'' makes the game easier for Carole, because the weight of each possible lie is reduced to w . Thus, if $n-m$ questions suffice to find the secret number in the game over the channel Γ'' starting from state σ^b , a fortiori $n-m$ questions will suffice in the game over the channel Γ' .

Let the encoding function θ send all elements of $\cup_{j=0}^{e-w} A_j$ one-one onto q -ary tuples in \mathcal{C}_1 , and all elements of $\cup_{j=e-w+1}^e A_j$ one-one onto q -ary tuples in \mathcal{C}_2 . Let $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ be the range of θ . Paul's second batch of questions will ask

What is the i th digit of the encoding $\theta(x_*)$? for $i = 1, \dots, n-m$.

Our constraints on the Hamming distance between any two q -tuples in \mathcal{C}_1 and \mathcal{C}_2 are just to ensure that Paul will be able to identify x_* , no matter Carole's mendacious answers. As a matter of fact, there are two possible cases:

Case 1. The secret number x_* is an element of $\cup_{j=0}^{e-w} A_j$.

Then Carole can give at most $\lfloor e/w \rfloor$ mendacious answers. Any tuple $\mathbf{x} \in \mathcal{C}$ representing a number $\neq x_*$ will differ from \mathbf{x}_*^θ in at least $2\lfloor e/w \rfloor + 1$ digits. From the tuple \mathbf{a} of Carole's answers, Paul will correctly guess x_* , as the only possible number corresponding to the tuple in \mathcal{C} at minimum distance from \mathbf{a} .

Case 2. The secret number belongs to $\cup_{j=e-w+1}^e A_j$.

Then Carole must give sincere answers to all remaining questions. As a matter of fact, each lie weighs at least w and, under our standing hypothesis for this case, Carole's available weight is less than w . Carole's answers will necessarily coincide with the encoding x_*^θ of x_* . Again, by choosing the tuple of \mathcal{C} which is closest to the tuple of Carole's answers, Paul will correctly guess the secret number.

In either case Paul must only find the tuple in \mathcal{C} which is closest to the tuple of Carole's answers. This tuple does correspond to Carole's secret number x_* . The proof of Theorem 3 is now complete.

Acknowledgments

The authors are grateful to Rudolf Ahlswede for the helpful discussions.

References

1. E. R. Berlekamp, Block coding for the binary symmetric channel with noiseless, delayless feedback. In: *Error-correcting Codes*, H.B. Mann (Editor), Wiley, New York (1968), pp. 61-88.
2. F. Cicalese, C. Deppe, Quasi-Perfect Minimally Adaptive q-ary Search with Unreliable Tests. *Proceedings of the 14th International Symposium, on Algorithms and Computation, ISAAC2003*, T. Ibaraki, N. Katoh, H. Ono (Eds.), *Lecture Notes in Computer Science* 2906, Springer-Verlag (2003), pp. 527-536.
3. F. Cicalese, C. Deppe, Q-ary Ulam-Rényi game with constrained lies. In: *General Theory of Information Transfer and Combinatorics*, R. Ahlswede et al., Ed., Shannon Foundation Publishers (2004), to appear.
4. F. Cicalese, D. Mundici, Optimal binary search with two unreliable tests and minimum adaptiveness. In: *Proc. European Symposium on Algorithms, ESA '99*, J. Nešetřil, Ed., *Lecture Notes in Computer Science* 1643, Springer-Verlag (1999), pp. 257-266.
5. F. Cicalese, D. Mundici, Perfect two fault-tolerant search with minimum adaptiveness. *Advances in Applied Mathematics*, 25 (2000), pp. 65-101.
6. F. Cicalese, D. Mundici, and U. Vaccaro, Least adaptive optimal search with unreliable tests. *Theoretical Computer Science*, 270 (2001), pp. 877-893.
7. F. Cicalese, D. Mundici, and U. Vaccaro, Rota-Metropolis cubic logic and Ulam-Rényi games. In: *Algebraic Combinatorics and Computer Science – A Tribute to Giancarlo Rota*, H. Crapo, D. Senato (Eds.), Springer-Verlag Italia, Milano (2001), pp. 197-244.
8. C. Deppe, Searching with lies and coding with feedback. In: *Search and Communication Complexity, Information Theory in Mathematics*, Series of Bolyai Studies, Springer-Verlag, Heidelberg, to appear.
9. I. Dumitriu and J. Spencer, The Liar Game over an Arbitrary Channel. (2003), to appear.
10. A. Pelc, Searching games with errors – fifty years of coping with liars. *Theoret. Comput. Sci.*, 270 (2002) 71–109.
11. A. Rényi, On a problem of information theory. *MTA Mat. Kut. Int. Kozl.*, 6B (1961), pp. 505–516.
12. S.M. Ulam, *Adventures of a Mathematician*. Scribner's, New York (1976).

Necessary and Sufficient Numbers of Cards for the Transformation Protocol (Extended Abstract)

Koichi Koizumi¹, Takaaki Mizuki^{2,3}, and Takao Nishizeki¹

¹ Graduate School of Information Sciences, Tohoku University,
Sendai 980-8579, Japan

`koizumi@nishizeki.ecei.tohoku.ac.jp`
`nishi@ecei.tohoku.ac.jp`

² Information Synergy Center, Tohoku University,
Sendai 980-8578, Japan

`mizuki@isc.tohoku.ac.jp`

³ PRESTO, JST, Saitama, 332-0012, Japan

Abstract. The transformation protocol can make two players share a secret key using a random deal of cards. A sufficient condition on the number of cards for the transformation protocol to succeed was known. However, it has been an open problem to obtain a necessary and sufficient condition. This paper improves the transformation protocol and gives a necessary and sufficient condition for the improved transformation protocol to succeed.

1 Introduction

A random deal of cards can be used for players to share a secret. For example, Winkler [13] gave bidding conventions for the game of bridge whereby one player can send secret information to her partner. This idea was carried further so that two players can share a secret key using a random deal of cards [1]. Since then, several protocols using a random deal of cards have been developed; Fischer and Wright gave two important protocols called the “key set protocol [2, 5]” and the “transformation protocol [3].” The properties of the key set protocol have been investigated extensively [6–12]. For instance, a necessary and sufficient condition on the number of cards for the key set protocol to succeed was known [7, 9]. On the other hand, concerning the transformation protocol, few results have been obtained so far. For instance, a sufficient condition for the transformation protocol to succeed was known [3]. However, it has been an open problem to obtain a necessary and sufficient condition. In this paper, we will address only the transformation protocol, and close the open problem above.

The scenario is as follows. Two players Alice and Bob communicate publicly, while a passive computationally-unlimited eavesdropper Eve overhears all communication. Alice and Bob are assumed to use randomization, that is, they can flip private fair coins. Let n be a positive real number such that $n = \log_2 m$ for some integer $m \geq 2$ and hence $m = 2^n$. Alice and Bob wish to share an n -bit

secret key $v \in \{1, 2, 3, \dots, 2^n (= m)\}$ which Eve cannot learn. That is, they wish to share a value $v \in \{1, 2, 3, \dots, 2^n\}$ such that, given the information available to Eve, the (conditional) probability of $v = \ell$ is exactly $1/2^n$ for every $\ell \in [1, 2^n]$.

Let $\Delta = \{1, 2, \dots, d\}$ be a *deck* of d distinct cards; an element in the deck Δ is a card. We call a subset $H \subseteq \Delta$ of Δ a *hand*. A sequence $\delta = (H_a, H_b; H_e)$ of three hands such that $\{H_a, H_b, H_e\}$ is a partition of Δ is called a *deal*. A deal $\delta = (H_a, H_b; H_e)$ means that every card in Δ is dealt to Alice, Bob or Eve so that Alice, Bob and Eve have hands H_a , H_b and H_e , respectively, as in the case of usual card games. We call $\gamma = (a, b; e)$ the *signature* of a deal $\delta = (H_a, H_b; H_e)$ if $a = |H_a|$, $b = |H_b|$ and $e = |H_e|$, where $|X|$ denotes the cardinality of a set X .

Fix a signature $\gamma = (a, b; e)$ with $a, b \geq 1$. For such γ , we always fix the deck $\Delta = \{1, 2, \dots, a + b + e\}$. Then, there are exactly $\binom{a+b+e}{a} \cdot \binom{b+e}{b}$ deals having the signature γ . Assume that Alice, Bob and Eve have their hands H_a , H_b and H_e , respectively, from a random deal $\delta = (H_a, H_b; H_e)$ whose signature is γ . As in the case of usual card games, all the cards in her/his hand are private to herself/himself. Given such a random deal δ , Alice and Bob wish to share a secret key: the goal is to design a protocol which makes Alice and Bob share an n -bit secret key that Eve cannot learn. We say that a protocol *establishes an n -bit secret key exchange* for a signature $\gamma = (a, b; e)$ if the protocol always makes Alice and Bob share an n -bit secret key $v \in \{1, 2, 3, \dots, 2^n\}$ for any random deal $\delta = (H_a, H_b; H_e)$ having the signature γ and any random result of flipping their coins.

In this paper, we first improve the transformation protocol. Our “improved transformation protocol” is superior to the transformation protocol. That is, the improved transformation protocol establishes an n -bit secret key exchange for any signature γ for which the (original) transformation protocol does. We then give a necessary and sufficient condition for the improved transformation protocol to establish an n -bit secret key exchange for a signature $\gamma = (a, b; e)$. We thus close the open problem above.

2 Preliminaries

In this section, we define some terms, and describe the transformation protocol [3] given by Fischer and Wright. Fix a signature $\gamma = (a, b; e)$ with $a, b \geq 1$, and let $\delta = (H_a, H_b; H_e)$ be a random deal having the signature γ .

A subset $S \subseteq \Delta$ of the deck Δ is called an (s, i, j) -*portion relative to δ* if $s = |S|$, $i, j \geq 1$, and S contains exactly i cards from Alice’s hand H_a , exactly j cards from Bob’s hand H_b and exactly $s - i - j$ cards from Eve’s hand H_e . We often omit the phrase “relative to δ ” if it is clear from the context. An (s, i, j) -portion S is said to be *complete* if $s = i + j$, i.e. Eve has no card in S . Furthermore, an (s, i, j) -portion S is said to be *partial* if $s > i + j$, i.e. Eve has at least one card in S . A portion S is said to be *opaque* if Eve does not know anything about the location of the cards in $S \setminus H_e$. Consider the case where Alice and Bob obtain an opaque complete (s, i, j) -portion S , i.e. an opaque $(i + j, i, j)$ -portion S . Since Eve has no card in S , Alice and Bob completely know the

owners of all cards in S , but Eve knows nothing about it. Therefore, from the portion S , Alice and Bob can share a $\log_2 \binom{i+j}{i}$ -bit secret key. Thus, an opaque complete portion immediately brings Alice and Bob a secret key.

A set \mathcal{C} of pairwise disjoint portions relative to δ is called a *collection relative to δ* . We often omit the phrase “relative to δ ” if it is clear from the context. We say that a collection $\mathcal{C} = \{S_1, S_2, \dots, S_m\}$ is *opaque* if Eve does not know anything about the location of the cards in $S_1 \setminus H_e, S_2 \setminus H_e, \dots, S_m \setminus H_e$. If Alice and Bob obtain an opaque collection \mathcal{C} containing complete portions, then they can share a secret key.

During any execution of the transformation protocol, Alice and Bob start with the *initial* collection $\mathcal{C}_0 = \{\Delta\}$, change \mathcal{C}_0 into another collection \mathcal{C}_1 , change \mathcal{C}_1 into \mathcal{C}_2 , and so on. They finally obtain an opaque *terminal* collection \mathcal{C}_t . A collection \mathcal{C}_ℓ can be changed into another collection $\mathcal{C}_{\ell+1}$, $0 \leq \ell \leq t-1$, by a *splitting transformation* or a *combining transformation*. A splitting transformation replaces an (s, i, j) -portion in the current collection with several smaller portions. A combining transformation replaces two $(s, 1, 1)$ -portions in the current collection with a single $(s', 1, 1)$ -portion for some $s' < s$.

To simplify the notation, we hereafter denote by \mathcal{C} the current collection which Alice and Bob maintain if it is clear from the context. Alice and Bob start with $\mathcal{C} = \mathcal{C}_0 = \{\Delta\}$. We sometimes use \mathcal{C}' to represent a collection resulting from the current collection \mathcal{C} by some transformation.

We first present how to apply a splitting transformation to \mathcal{C} , i.e. how to split a portion S in \mathcal{C} .

Splitting: An (s, i, j) -portion S with $i + j \geq 3$ can be split so that several smaller new portions will be acquired. If $i \geq j$, then the splitting transformation proceeds as described below. If $i < j$, then the roles of Alice and Bob are reversed.

1. Alice randomly partitions S into i sets S'_1, S'_2, \dots, S'_i , each of size $\lfloor s/i \rfloor$ or $\lceil s/i \rceil$, such that she has exactly one card in each set, and announces the sets.
2. Bob says how many cards he has in each set announced by Alice.
3. Each set in which Bob has at least one card is acquired as a new portion.

Notice that any $(s, 1, 1)$ -portion cannot be split. For an (s, i, j) -portion S , we say that S is *splittable* if $i + j \geq 3$; and S is *non-splittable* if $i + j = 2$, i.e. $i = j = 1$.

Alice and Bob repeat applying a splitting transformation to \mathcal{C} until any portion in \mathcal{C} cannot be split. Then each portion S_ℓ , $1 \leq \ell \leq m$, in the current collection $\mathcal{C} = \{S_1, S_2, \dots, S_m\}$ is a $(|S_\ell|, 1, 1)$ -portion. They next repeat applying the following combining transformation to \mathcal{C} , i.e. repeat combining two portions S_1 and S_2 having the same sizes in \mathcal{C} .

Combining: Two $(s, 1, 1)$ -portions S_1 and S_2 with $s \geq 3$ can be combined so that a new portion S' will be acquired.

1. Alice randomly chooses an integer $p \in \{1, 2\}$. Let $q = 3 - p$.
2. Alice constructs and announces a set T consisting of her card in S_p , $\lfloor s/3 \rfloor - 1$ cards randomly chosen from S_p that are not hers, and $\lfloor s/3 \rfloor$ cards randomly chosen from S_q that are not hers.

3. Bob announces how many cards he has in T .
 - (a) If Bob has no card in T , then Alice announces the set difference $S_q \setminus T$, which is acquired as a new portion $S' = S_q \setminus T$.
 - (b) If Bob has exactly one card in T , then T is acquired as a new portion $S' = T$.
 - (c) If Bob has two cards in T , then Alice announces $S_p \cap T$, which is acquired as a new portion $S' = S_p \cap T$.

As Alice and Bob repeat combining two portions in the current collection \mathcal{C} , the sizes of the portions in \mathcal{C} become smaller. Notice that $(2, 1, 1)$ -portions cannot be combined. When they cannot apply a combining transformation to \mathcal{C} , they obtain a terminal collection $\mathcal{C} = \mathcal{C}_t$; the terminal collection \mathcal{C}_t possibly contains $(2, 1, 1)$ -portions, which can be used to share a secret key.

We are now ready to give the full description of the transformation protocol. Given a random deal $\delta = (H_a, H_b; H_e)$, the transformation protocol proceeds as follows.

Transformation protocol:

1. The initial collection is $\mathcal{C} = \mathcal{C}_0 = \{\Delta\}$.
2. Splitting is repeated as long as there is a splittable portion, i.e. an (s, i, j) -portion with $i + j \geq 3$, in \mathcal{C} : choose such a portion S in \mathcal{C} according to any prearranged rule, remove S from \mathcal{C} , apply a splitting transformation to S , and add all the new acquired portions to \mathcal{C} .
3. Combining is repeated as long as there is a pair of $(s, 1, 1)$ -portions with $s \geq 3$ in \mathcal{C} : choose such two portions S_1 and S_2 in \mathcal{C} according to any prearranged rule, remove both S_1 and S_2 from \mathcal{C} , apply a combining transformation to S_1 and S_2 , and add the new acquired portion to \mathcal{C} .
4. From the terminal collection $\mathcal{C} = \mathcal{C}_t$, Alice and Bob share an n -bit secret key, where n is the number of $(2, 1, 1)$ -portions in \mathcal{C}_t .

We now describe the definitions of the “potential function” ϕ and the constant W [3]. Let $c = \log_{3/2} 2$. The potential function $\phi(s, i, j)$ is recursively defined as follows:

$$\phi(s, i, j) = \begin{cases} 2 & \text{if } s = 2 \text{ and } i = j = 1; \\ (s - 2)^{-c} & \text{if } s \geq 3 \text{ and } i = j = 1; \\ j\phi(\lceil s/i \rceil, 1, 1) & \text{if } i \geq j \text{ and } i \geq 2; \text{ and} \\ \phi(s, j, i) & \text{if } i < j. \end{cases}$$

The constant W is defined as $W = \sum_{s=3}^{\infty} (s - 2)^{-c}$. (One can show that $2.0356 < W < 2.0358$ [3].) Using ϕ and W , we present the sufficient condition given by Fischer and Wright as in the following Theorem 1.

Theorem 1 ([3]) *Let n be a positive integer, and let $\gamma = (a, b; e)$ be a signature with $a, b \geq 1$. If $\phi(a + b + e, a, b) > W + 2(n - 1)$, then the transformation protocol establishes an n -bit secret key exchange for γ .*

The condition in Theorem 1, i.e. $\phi(a + b + e, a, b) > W + 2(n - 1)$, is a sufficient condition for the transformation protocol to establish an n -bit secret

key exchange for $\gamma = (a, b; e)$. However, it is not a necessary condition in general. It has been an open problem to obtain a necessary and sufficient condition. This paper closes the open problem as in the succeeding section.

3 Improved Transformation Protocol

In this section, we first slightly modify the transformation protocol, and then give a necessary and sufficient condition for our improved transformation protocol to establish an n -bit secret key exchange for a signature $\gamma = (a, b; e)$. There are two main ideas behind the modification. In the remainder of this paper, all logarithms are to the base 2.

3.1 Stopping Useless Splitting Transformations

In this subsection, we explain the first improvement, i.e. stopping a “useless” splitting transformation; the idea behind the improvement is naive.

We now explain a “useless” splitting transformation, as follows. Assume that, during the execution of the transformation protocol, Alice and Bob obtain an opaque complete splittable portion, say an opaque $(3, 1, 2)$ -portion S . According to the transformation protocol, S is eventually split into a $(2, 1, 1)$ -portion S'_1 and a singleton set S'_2 consisting of one card from Bob’s hand. Since the singleton set S'_2 contains no Alice’s card, S'_2 is discarded, and hence only the $(2, 1, 1)$ -portion S'_1 is acquired as a new portion. Comparing the original $(3, 1, 2)$ -portion S and the acquired $(2, 1, 1)$ -portion S'_1 , S is preferable to S'_1 , because a $\log 3$ -bit secret key is distilled from S while only a one-bit secret key is distilled from S'_1 . Thus, it is “useless” to split S . More generally, one can immediately notice that it is *useless* to split a complete portion, i.e. an $(i + j, i, j)$ -portion for some i and j , which can be used to share a $\log \binom{i+j}{i}$ -bit secret key. Therefore, we never split a complete portion in our transformation protocol. This is the idea behind the first improvement. The full description of our transformation protocol will be given in Section 3.3.

In the remainder of this subsection, we introduce two functions ψ_C and ψ_P , which will be used later to describe a necessary and sufficient condition for our transformation protocol to establish an n -bit secret key exchange for a signature $\gamma = (a, b; e)$.

We first introduce a function ψ_C which maps a portion S (relative to a deal δ) to a nonnegative real number. (Strictly speaking, the variable of the function ψ_C should be a pair (S, δ) instead of S , but we write simply $\psi_C(S)$.) The function ψ_C is called the *completely potential function*. Intuitively, it means that a $\psi_C(S)$ -bit secret key can be distilled directly from a portion S . Remember that, from a complete portion, namely an $(i + j, i, j)$ -portion, Alice and Bob can share a $\log \binom{i+j}{i}$ -bit secret key. Hence, we define the completely potential function $\psi_C(S) = \psi_C(s, i, j)$ for an (s, i, j) -portion S as follows:

$$\psi_C(s, i, j) = \begin{cases} \log \binom{i+j}{i} & \text{if } s = i + j; \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

We next introduce a function ψ_P which maps a portion S (relative to a deal δ) to a nonnegative real number. The function ψ_P is called the *partially potential function*. Roughly speaking, whereas a $\psi_C(S)$ -bit secret key can always be distilled directly from a portion S , a $\psi_P(S)$ -bit secret key may be distilled from S by splitting and/or combining.

We first define the partially potential function $\psi_P(S) = \psi_P(s, i, j)$ for a complete portion S , i.e. an (s, i, j) -portion S with $s = i + j$. Since such a portion S is never split and never combined, we define

$$\psi_P(s, i, j) = 0 \quad \text{if } s = i + j. \quad (2)$$

We then define ψ_P for a partial portion S , i.e. an (s, i, j) -portion S with $s > i + j$. We will define ψ_P for a partial non-splittable portion S in the succeeding subsection; thus, we now consider a partial splittable portion S , i.e. an (s, i, j) -portion S such that $s > i + j \geq 3$. Note that such a portion S will be split. We recursively set $\psi_P(S)$ to be the summation of $\psi_P(S'_1), \psi_P(S'_2), \dots, \psi_P(S'_m)$, where S'_1, S'_2, \dots, S'_m are the portions acquired by some particular splitting transformation, called the “worst” splitting. Assume for the moment that $i \geq j$, i.e. Alice does not have fewer cards in S than Bob. Furthermore, assume that Alice does not have more cards in S than Eve, i.e. $s - (i + j) \geq i$, or $s \geq 2i + j$. As the “worst” splitting, we consider the case where Alice would split the portion S into i subportions so that Bob has exactly one card in every S'_ℓ , $1 \leq \ell \leq j$. One may assume that S'_1, S'_2, \dots, S'_i are sorted in non-increasing order of their cardinalities. Note, in this case, that each of the acquired portions S'_1, S'_2, \dots, S'_j is either a $(\lceil s/i \rceil, 1, 1)$ -portion or a $(\lfloor s/i \rfloor, 1, 1)$ -portion. Note, furthermore, that each of the first j subportions S'_1, S'_2, \dots, S'_j is a partial portion, i.e. $|S'_\ell| \geq 3$ for every ℓ , $1 \leq \ell \leq j$, because Eve has i or more cards. We now count the numbers of $(\lceil s/i \rceil, 1, 1)$ -portions and $(\lfloor s/i \rfloor, 1, 1)$ -portions. Let r be the remainder when dividing s by i , that is, let $r = s \bmod i$. If $r = 0$, then there are exactly j $(s/i, 1, 1)$ -portions. If $1 \leq r < j$, then there are exactly r $(\lceil s/i \rceil, 1, 1)$ -portions and exactly $j - r$ $(\lfloor s/i \rfloor, 1, 1)$ -portions. If $j \leq r$, then there are exactly j $(\lceil s/i \rceil, 1, 1)$ -portions. Thus, for an (s, i, j) -portion S such that $s > i + j \geq 3$, $i \geq j$ and $s \geq 2i + j$, we define

$$\psi_P(s, i, j) = \begin{cases} j\psi_P(s/i, 1, 1) & \text{if } r = 0; \\ r\psi_P(\lceil s/i \rceil, 1, 1) + (j - r)\psi_P(\lfloor s/i \rfloor, 1, 1) & \text{if } 1 \leq r < j; \\ j\psi_P(\lceil s/i \rceil, 1, 1) & \text{if } j \leq r, \end{cases} \quad (3)$$

where $r = s \bmod i$. Next assume that Alice has more cards than Eve, i.e. $s - (i + j) < i$, or $s < 2i + j$. Then, at least one $(2, 1, 1)$ -portion is always produced, and hence, for an (s, i, j) -portion S such that $s > i + j \geq 3$, $i \geq j$ and $s < 2i + j$, we define

$$\psi_P(s, i, j) = 1 \quad \text{if } s > i + j \geq 3, i \geq j \text{ and } s < 2i + j. \quad (4)$$

For the case of $i < j$, we define

$$\psi_P(s, i, j) = \psi_P(s, j, i) \quad \text{if } s > i + j \geq 3 \text{ and } i < j. \quad (5)$$

3.2 Combining with Dummy Cards

In this subsection, we explain the second improvement; we introduce an operation called “combining with dummy cards,” whereby Alice and Bob can efficiently utilize “unused” portions.

We first explain an *unused* portion. Consider the case where Alice and Bob obtain a terminal collection $\mathcal{C}_t = \{S_1, S_2, \dots, S_m\}$ when the transformation protocol terminates. There is no pair of portions S_g and S_ℓ in \mathcal{C}_t with $s_g = s_\ell \geq 3$. If all the m portions in \mathcal{C}_t are $(2, 1, 1)$ -portions, then Alice and Bob share an m -bit secret key, and hence there is no “unused” portion. However, if \mathcal{C}_t contains an $(s, 1, 1)$ -portion with $s \geq 3$, then such a portion is not used to share a secret key, and hence it is unused.

In order to utilize unused portions, we need to combine two portions of different sizes. For this purpose, we add “dummy” cards to the smaller portion. For example, consider a $(6, 1, 1)$ -portion S_1 and a $(5, 1, 1)$ -portion S_2 . We add one *dummy* card x to the portion S_2 so that the resulting portion $U_2 = S_2 \cup \{x\}$ has the same size as S_1 . The dummy card x is chosen not in $S_1 \cup S_2$. Alice and Bob regard the dummy card x as Eve’s card. Then, we apply to S_1 and U_2 a combining transformation described in Section 2. Let U' be the new portion acquired by combining. If U' has the dummy card x , then remove it from U' . That is, let $S' = U' \setminus \{x\}$, which is acquired as a new portion. In this way, one can combine two portions of different sizes. We thus obtain the following operation, called *combining with dummy cards*.

Combining with dummy cards: An $(s_1, 1, 1)$ -portion S_1 and an $(s_2, 1, 1)$ -portion S_2 with $s_1 \geq s_2 \geq 3$ can be combined so that a new portion S' will be acquired, as follows.

1. Let D be any set of dummy cards such that $|D| = s_1 - s_2$ and $D \cap (S_1 \cup S_2) = \emptyset$. All the dummy cards in D are added to S_2 , that is, let $U_1 = S_1$ and $U_2 = S_2 \cup D$. Note that D is an empty set if $s_1 = s_2$.
2. Alice randomly chooses an integer $p \in \{1, 2\}$. Let $q = 3 - p$.
3. Alice constructs and announces a set T consisting of her card in U_p , $\lfloor s_1/3 \rfloor - 1$ cards randomly chosen from U_p that are not hers, and $\lfloor s_1/3 \rfloor$ cards randomly chosen from U_q that are not hers.
4. Bob announces how many cards he has in T .
 - (a) If Bob has no cards in T , then Alice announces the set difference $U_q \setminus T$ and let $U' = U_q \setminus T$.
 - (b) If Bob has exactly one card in T , then let $U' = T$.
 - (c) If Bob has two cards in T , then Alice announces $U_p \cap T$ and let $U' = U_p \cap T$.
5. If U' has dummy cards, then remove them from U' , i.e. let $S' = U' \setminus D$. Alice and Bob acquire S' as a new portion.

When an $(s_1, 1, 1)$ -portion S_1 and an $(s_2, 1, 1)$ -portion S_2 such that $s_1 \geq s_2$ are combined with dummy cards, the acquired portion S' has size at most $\lfloor 2s_1/3 \rfloor$; in particular, if Alice chooses $p = 2$ in step 2 and Bob has no card in T

announced by Alice, then the acquired portion $S' = (U_1 \setminus T) \setminus D = S_1 \setminus T$ contains no dummy card, and hence $|S'| = \lceil 2s_1/3 \rceil$. Thus, in the “worst” combining, the acquired portion S' always has the size of exactly $\lceil 2s_1/3 \rceil$.

Since we have not defined ψ_P for a partial non-splittable portion, i.e. an (s, i, j) -portion with $s > i + j = 2$, we complete the definition of ψ_P in the remainder of this subsection. That is, we define $\psi_P(s, 1, 1)$ for $s \geq 3$. Note that, by combining, two $(3, 1, 1)$ -portions become a $(2, 1, 1)$ -portion which yields a one-bit secret key, and that two $(s, 1, 1)$ -portions with $s \geq 4$ become a $(\lceil 2s/3 \rceil, 1, 1)$ -portion in the “worst” case. Thus, we recursively define

$$\begin{cases} \psi_P(3, 1, 1) = 1/2; \text{ and} \\ \psi_P(s, 1, 1) = \frac{1}{2} \psi_P(\lceil 2s/3 \rceil, 1, 1) \text{ if } s \geq 4. \end{cases} \quad (6)$$

Notice that $\psi_P(s, 1, 1)$ is monotonically decreasing in s for all integers $s \geq 3$. We have thus completed the definition of ψ_P . In our transformation protocol whose full description will appear in the succeeding subsection, we combine an $(s_1, 1, 1)$ -portion S_1 and an $(s_2, 1, 1)$ -portion S_2 with dummy cards only if $\psi_P(S_1) = \psi_P(S_2)$; otherwise, the “partially potential” may decrease; for example, if a $(3, 1, 1)$ -portion S_1 and a $(4, 1, 1)$ -portion S_2 were combined, then a $(3, 1, 1)$ -portion S' would be obtained in the “worst” case, and hence the “partially potential” decreases by $1/2^2$.

3.3 Our Protocol and Results

We generalize a key set protocol [2, 5] to a “multiple key sets protocol,” whose definition is omitted in this extended abstract due to page limitation. As explained in Sections 3.1 and 3.2, we modify the transformation protocol and obtain the following *improved transformation protocol*. Given a random deal δ , the improved transformation protocol proceeds as follows.

Improved transformation protocol:

1. If the signature $\gamma = (a, b; e)$ of δ satisfies $0 < e < \max\{a, b\}$, then the multiple key sets protocol is executed so that Alice and Bob share at least a $\min\{a, b, \lfloor (a + b - e)/2 \rfloor\}$ -bit secret key, and the improved transformation protocol terminates. If $e = 0$ or $e \geq \max\{a, b\}$, then go to step 2.
2. The initial collection is $\mathcal{C} = \mathcal{C}_0 = \{\Delta\}$.
3. Splitting is repeated as long as \mathcal{C} contains a partial splittable portion, i.e. an (s, i, j) -portion with $s > i + j \geq 3$: choose such a portion S in \mathcal{C} according to any prearranged rule, remove S from \mathcal{C} , apply a splitting transformation to S , and add all the new acquired portions to \mathcal{C} .
4. The operation of the combining with dummy cards is repeated as long as there are two portions S_1 and S_2 in \mathcal{C} such that S_1 is an $(s_1, 1, 1)$ -portion, S_2 is an $(s_2, 1, 1)$ -portion, $s_1 \geq s_2 \geq 3$, and $\psi_P(S_1) = \psi_P(S_2)$: choose such two portions S_1 and S_2 in \mathcal{C} according to any prearranged rule, remove both S_1 and S_2 from \mathcal{C} , apply a combining transformation with dummy cards to S_1 and S_2 , and add the new acquired portion to \mathcal{C} .

5. Alice and Bob share a $\sum_{S \in \mathcal{C}_t} \psi_{\mathcal{C}}(S)$ -bit secret key from the terminal collection $\mathcal{C} = \mathcal{C}_t$, and the improved transformation protocol terminates.

In step 1, we use the multiple key sets protocol because this protocol is effective when a signature $\gamma = (a, b; e)$ satisfies $0 < e < \max\{a, b\}$.

We now give the definition of our potential function ψ which maps a collection \mathcal{C} to a nonnegative real number as follows:

$$\psi(\mathcal{C}) = \sum_{S \in \mathcal{C}} \psi_{\mathcal{C}}(S) + \left\lfloor \sum_{S \in \mathcal{C}} \psi_{\mathcal{P}}(S) \right\rfloor. \quad (7)$$

(We take the floor in the second term of the right hand side, because a set of partial portions is transformed into several $(2, 1, 1)$ -portions, each of which yields a one-bit secret key.) We write $\psi(a + b + e, a, b)$ instead of $\psi(\mathcal{C}_0)$ if $\mathcal{C}_0 = \{\Delta\}$ is a singleton collection relative to a deal δ having a signature $\gamma = (a, b; e)$.

Considering the case where the multiple key sets protocol runs, we define a function $\Psi(a, b; e)$ for a signature $\gamma = (a, b; e)$ with $a, b \geq 1$, as follows:

$$\Psi(a, b; e) = \begin{cases} \min\{a, b, \lfloor (a + b - e)/2 \rfloor\} & \text{if } 0 < e < \max\{a, b\}; \\ \psi(a + b + e, a, b) & \text{otherwise.} \end{cases} \quad (8)$$

We have the following Theorem 2 as our main result. We omit a proof of Theorem 2 due to the page limitation.

Theorem 2 *Let $n = \log m$ for an integer $m \geq 2$, and let $\gamma = (a, b; e)$ be a signature with $a, b \geq 1$. Then the improved transformation protocol establishes an n -bit secret key exchange for γ if and only if $\Psi(a, b; e) \geq n$.*

4 Conclusions

The transformation protocol can efficiently make players share a perfect secret key using a random deal of cards. A sufficient condition for the transformation protocol to establish an n -bit secret key exchange was known. However, it has been an open problem to obtain a necessary and sufficient condition. This paper improves the transformation protocol, and gives a necessary and sufficient condition for the improved transformation protocol to establish an n -bit secret key exchange as in Theorem 2. Our improved transformation protocol is entirely superior to the original transformation protocol. Thus, Theorem 2 closes the open problem above.

Fischer and Wright [3, 5] proposed a method for reducing the problem of a multiparty n -bit secret key exchange to the problem of a 2-party n -bit secret key exchange. Hence, using this method, one can easily extend our protocol so that it performs a k -party n -bit secret key exchange with $k \geq 3$.

This paper addresses only the transformation protocol. Therefore, it still remains open to obtain a necessary and sufficient condition for any (not necessarily transformation) protocol to establish an n -bit secret key exchange for a signature γ [4, 14].

References

1. M. J. Fischer, M. S. Paterson, and C. Rackoff, "Secret bit transmission using a random deal of cards," DIMACS Series in Discrete Mathematics and Theoretical Computer Science, AMS, vol. 2, pp. 173–181, 1991.
2. M. J. Fischer and R. N. Wright, "An application of game-theoretic techniques to cryptography," DIMACS Series in Discrete Mathematics and Theoretical Computer Science, AMS, vol. 13, pp. 99–118, 1993.
3. M. J. Fischer and R. N. Wright, "An efficient protocol for unconditionally secure secret key exchange," Proc. the 4th Annual Symposium on Discrete Algorithms, pp. 475–483, 1993.
4. M. J. Fischer and R. N. Wright, "Bounds on secret key exchange using a random deal of cards," J. Cryptology, vol. 9, pp. 71–99, 1996.
5. M. J. Fischer and R. N. Wright, "Multiparty secret key exchange using a random deal of cards," Proc. CRYPTO '91, Lecture Notes in Computer Science, vol. 576, pp. 141–155, 1992.
6. T. Mizuki and T. Nishizeki, "Necessary and sufficient numbers of cards for sharing secret keys on hierarchical groups," IEICE Trans. Inf. & Syst., vol. E85-D, no. 2, pp. 333–345, 2002.
7. T. Mizuki, H. Shizuya, and T. Nishizeki, "A complete characterization of a family of key exchange protocols," International Journal of Information Security, vol. 1, no. 2, pp. 131–142, 2002.
8. T. Mizuki, H. Shizuya, and T. Nishizeki, "Characterization of optimal key set protocols," Discrete Applied Mathematics, vol. 131, pp. 213–236, 2003.
9. T. Mizuki, H. Shizuya, and T. Nishizeki, "Dealing necessary and sufficient numbers of cards for sharing a one-bit secret key," Proc. EUROCRYPT '99, Lecture Notes in Computer Science, vol. 1592, pp. 389–401, 1999.
10. T. Mizuki, H. Shizuya, and T. Nishizeki, "Eulerian secret key exchange," Proc. COCOON '98, Lecture Notes in Computer Science, vol. 1449, pp. 349–360, 1998.
11. T. Mizuki, Z. Sui, H. Shizuya, and T. Nishizeki, "On the average length of secret key exchange Eulerian circuits," IEICE Trans. Fundamentals, vol. E83-A, no. 4, pp. 662–670, 2000.
12. R. Yoshikawa, S. Guo, K. Motegi, and Y. Igarashi, "Construction of secret key exchange spanning trees by random deals of cards on hierarchical structures," IEICE Trans. Fundamentals, vol. E84-A, no. 5, pp. 1110–1119, 2001.
13. P. Winkler, "The advent of cryptology in the game of bridge," CRYPTOLOGIA, vol. 7, pp. 327–332, 1983.
14. R. N. Wright, "Achieving Perfect Secrecy Using Correlated Random Variables," PhD Thesis, Yale University, 1994.

On the Selection and Assignment with Minimum Quantity Commitments

Andrew Lim, Fan Wang, and Zhou Xu*

Dept Industrial Engineering and Engineering Management,
Hong Kong University of Science and Technology,
Clear Water Bay, Kowloon, Hong Kong
{iealim,fanwang,xuzhou}@ust.hk

1 Introduction and Problem Statement

Traditional selection and assignment problems analyze how to select and assign the suppliers to meet demands within minimum total costs. The maximum capacity and the fixed selection costs are two major restrictions imposed on suppliers, and extensively studied in literature.

In practice, another constraint, named as Minimum Quantity Commitment (MQC), needs to be satisfied by each selected supplier. It is stipulated for the global transportation by the US Federal Maritime Commission. According to the MQC, when assigning containers for shipments to US, the international companies, including the Royal Philips Electronics Company, must insure each selected shipping agent to have a certain minimum quantity of containers. This increases difficulties for them to seek the optimal schedule with minimum total costs. The MQC constraint has been seldom studied before, which motivates us to present its hardness results for the selection and assignment problems.

Before formulating the new MQC constraint, we review some traditional problems first. Let $J = \{1, \dots, n\}$ denote the set of transportation requests, each requiring d_j ($j \in J$) containers to be shipped from a source to a destination. Let $I = \{1, \dots, m\}$ denote the set of shipping agents, and each agent $i \in I$ bids a transportation cost, denoted by $c_{i,j}$, for shipping a container in request j . The *basic selection and assignment problem* can be defined to select a subset $A \subseteq I$ of shipping agents, and to assign every $i \in A$ a proper number of containers, say $z_{i,j}$, for each request $j \in J$, so that the following total cost is minimized:

$$\sum_{i \in A} \sum_{j \in J} c_{i,j} z_{i,j}, \quad (1)$$

while the demand constraint is satisfied:

$$\sum_{i \in A} z_{i,j} = d_j, \quad (2)$$

This basic problem can be easily solved by a greedy approach, that is to assign each request for container shipment to the lowest bidder or agent. However, it

* Corresponding Author

becomes more difficult, if the maximum capacity, s_i , for each agent $i \in I$ is considered, which leads the capacity constraint:

$$\sum_{j \in J} z_{i,j} \leq s_i. \quad (3)$$

The capacitated variant above is known as the *transportation problem* [1], which can not be solved by the simple greedy method, but can be transformed to a minimum cost flow problem and is still polynomial solvable [1].

Besides the transportation cost, the fixed selection cost f_i for selecting each agent $i \in A$ is often considered in literature as well. It extends the objective functions to be:

$$\sum_{i \in A} f_i + \sum_{i \in A} \sum_{j \in J} c_{i,j} z_{i,j}. \quad (4)$$

The fixed selection cost makes the problem intractable. No matter whether the capacity constraint (3) is considered (or not), the selection and assignment problem with fixed selection costs, called *uncapacitated (or capacitated resp.) facility location problem* [2, 6], is unary *NP*-hard [5]. However, being part of the objective function, the fixed selection cost does not affect the search for feasible solutions. As shown in Section 3.2, we can easily generate feasible solutions to satisfy the demand constraint (2) and/or the maximum capacity constraint (3).

However, finding feasible solutions might become intractable if the MQC constraint is considered. We generalize the MQC constraint to w regions. Let $R = \{1, \dots, w\}$ denote the set of regions with MQC constraint. For each region $r \in R$, its minimum quantity is b_r and the set of transportation requests with destinations in r is $J_r \subseteq J$. It is reasonable to assume that J_p and J_q are disjoint for any two different regions p and q in R . Therefore, the MQC constraint can be modeled as:

$$\sum_{j \in J_r} z_{i,j} = 0, \text{ or } \sum_{j \in J_r} z_{i,j} \geq b_r \quad (5)$$

Hence, the selection and assignment problem with the MQC constraint is to minimize the total cost (4) subjected to the demand constraint (2), the maximum capacity constraint (3) and the new MQC constraint (5). Its solution, represented by the selected agent set A and the assignments $\{z_{i,j}\}$, is feasible if and only if it satisfies the above three constraints, and is optimal if and only if it is a feasible solution and has the minimum total cost (4). The problem to find an optimal solution is the optimization variant of this problem. Similarly, the problem of finding a feasible solution is the feasibility variant of this problem.

Our paper presents the computational complexity results of finding an optimal (or feasible) solution for the selection and assignment problem with minimum quantity commitments. We study various special cases with different specifications of maximum capacity s_i , fixed selection cost f_i , and number of MQC regions w . For each special case, we will either give a polynomial algorithm, or

prove its binary \mathcal{NP} -hardness [5] with a pseudo-polynomial algorithm, or prove its unary \mathcal{NP} -hardness [5]. Our result reveals how the new MQC constraint affects the hardness of the selection and assignment problem. Due to space limitation, details of some proofs are left out here, but can be found in the long paper [8].

To make consistent the notations throughout the paper, let J_0 be the set of transportation requests not subjected to the MQC constraint, and fix its minimum quantity b_0 to be zero. Let $U = R \cup \{0\}$ denote the universal set, where region 0 is the complement of all the regions with MQC restriction. Let $D = \sum_{j \in J} d_j$ denote the total demand of transportation requests and $S = \sum_{i \in I} s_i$ denotes the total maximum capacity of shipping agents. These two totals often appear in this paper.

2 Classifications

To help with our classification, a four-field notation $Obj(\alpha, \beta, \gamma)$ is proposed here, where Obj defines the objective (optimization or feasibility), α indicates the number of regions with the MQC constraint, β gives the maximum capacity for agents, and γ describes the fixed selection cost.

The field Obj is the objective function. It has two possibilities:

- *Optimization*: the objective is to find the optimal solution;
- *Feasibility*: the objective is relaxed, we only want to find a feasible solution.

The field α indicates the number of regions subjected to MQC constraint. α has the following possibilities:

- w : the MQC region number is arbitrary and specified as a part of the problem's input;
- $w = \lambda$: the MQC region number is a constant, and is excluded from the problem's input;
- $w = 0, 1, 2$, or etc.: the MQC region number is a given integer. For example, if α is equal to $w = 1$, only one region has the MQC constraint.

The field β describes the maximum capacities of agents. β can take the following values:

- s_i : the maximum capacities of agents are arbitrary;
- $s_i = \infty$: the maximum capacities of agents are all infinity, i.e. the maximum capacity restriction (3) is relaxed.

Lastly, the field γ describes the fixed selection costs of agents, γ can be:

- f_i : the fixed selection costs of agents are arbitrary;
- $f_i = 0$: the fixed selection costs of agents are all zero, i.e. the total cost of a solution is relaxed from (4) to the total transportation cost (1) only.

3 Computational Complexity

The main complexity results are summarized in Table 1. From the results, we can derive a complete map of the computational complexity results for all special cases by their reduction relations shown in Section 2. In Table 1, we use Unary \mathcal{NP} -hard and Binary \mathcal{NP} -hard, defined in [5], to express that a problem is \mathcal{NP} -hard with respect to a unary and binary encoding of the data respectively. Along with each result, we provide a reference where its proof can be found. The entries “*” in the γ column of the the *Feasibility* rows imply that those results hold true for any specification of the fixed selection costs (f_i or $f_i = 0$), because the total cost (4) can be neglected when a feasible solution is considered. The complexity results show that

1. finding an optimal solution is unary \mathcal{NP} -hard unless both MQC restrictions and fixed selection costs are relaxed, i.e., $w = 0$ and $f_i = 0$, and
2. finding only a feasible solution is at least binary \mathcal{NP} -hard unless at most one region holds the MQC rule or the maximum capacity is relaxed, i.e., $w \leq 1$ or $s_i = \infty$.

Table 1. The Complexity Results

Objective	α	β	γ	Complexity Results	
Optimization	$w = 0$	$s_i = \infty$	$f_i = 0$	$O(nm)$	Thm 1
	$w = 0$	s_i	$f_i = 0$	$O((n + m)^4 \log(n + m))$	[7]
	$w = 0$	$s_i = \infty$	f_i	Unary \mathcal{NP} -hard	[4]
	$w = 1$	$s_i = \infty$	$f_i = 0$	Unary \mathcal{NP} -hard	Thm 4
Feasibility	w	$s_i = \infty$	*	$O(nm)$	Thm 5
	$w = 0$	s_i	*	$O(nm)$	Thm 6
	$w = 1$	s_i	*	$O(nm)$	Thm 9
	$w = 2$	s_i	*	Binary \mathcal{NP} -hard $O(mS^6 + nm)$	Thm 8 Cor 1
	$w = \lambda$	s_i	*	Binary \mathcal{NP} -hard $O(mS^{2\lambda+2} + nm)$	Cor 2 Thm 7
	w	s_i	*	Unary \mathcal{NP} -hard	Thm 10

3.1 Complexity of Finding an Optimal Solution

We begin with the case *Optimization*($w = 0, s_i = \infty, f_i = 0$), which can be reduced to all other optimization cases by the reduction relations shown in Section 2. For this case, both the maximum capacity constraint (3) and the MQC constraint (5) are relaxed. The absence of the fixed selection cost allows us to select all the agents without penalty. Therefore, we can assign each transportation request greedily to the agent with the lowest bid for that request. It is easy to see the total time complexity is $O(mn)$, leading the following result.

Theorem 1. *Optimization($w = 0, s_i = \infty, f_i = 0$) can be solved in $O(nm)$ time.*

The next case $Optimization(w = 0, s_i, f_i = 0)$ is equivalent to the transportation problem [1]. As we mentioned in Section 1, it can be solved by the minimum cost flow algorithm presented in [7]:

Theorem 2. *The minimum cost flow algorithm generates an optimal solution for the case $Optimization(w = 0, s_i, f_i = 0)$ in $O((n + m)^4 \log(n + m))$ time.*

We now consider the case $Optimization(w = 0, s_i = \infty, f_i)$ that has fixed selection costs f_i without the maximum capacity constraint and the MQC constraint. Since this case is equivalent to the uncapacitated facility location problem [4], a well-known unary \mathcal{NP} -hard problem, we have the following result:

Theorem 3. *Finding an optimal solution for the case $Optimization(w = 0, s_i = \infty, f_i)$ is unary \mathcal{NP} -hard.*

Next we consider the case $Optimization(w = 1, s_i = \infty, f_i = 0)$ that has one the MQC constraint for one region but relaxes the maximum capacity constraint and ignores the fixed selection costs. It can be proved to be unary \mathcal{NP} -hard by a reduction from **Cover By 3-Sets (X3C)** problem [5]. (See [8].)

Theorem 4. *Finding an optimal solution for the case $Optimization(w = 1, s_i = \infty, f_i = 0)$ is unary \mathcal{NP} -hard.*

3.2 Complexity of Finding a Feasible Solution

Feasibility cases are studied in this part. It needs to consider the constraints (2) (3) (5) but not the total cost (4). So, let the field γ be “*”, as the fixed selection cost does not change the complexity results. Therefore, we can select all the agents safely by assuming $A = \{1, 2, \dots, m\}$, and concentrate our study on how to obtain the assignment $z_{i,j}$ for $i \in A$ and $j \in J$.

For uncapacitated cases with $s_i = \infty$, let us look at the most general case $Feasibility(w, s_i = \infty, *)$. Its feasible assignment can be constructed easily by allocating all the transportation requests with full demands to a certain agent. The process is described in Algorithm 1, whose correctness can be easily seen.

Theorem 5. *Algorithm 1 generates a feasible solution for the case $Feasibility(w, s_i = \infty, *)$ in $O(nm)$ time.*

Algorithm 1 Solving $Feasibility(w, s_i = \infty, *)$

```

1: if for some region  $r \in R$ ,  $b_r$  exceed the total demands  $D$  of requests in region  $r$  then
2:   Output “No Feasible Solution”;
3: else
4:   Initially,  $z_{i,j} \leftarrow 0$  for all  $i \in I$  and  $j \in J$ ;
5:   Select all the shipping agents:  $A \leftarrow \{1, 2, \dots, m\}$ ;
6:   for all the transportation request  $j \in J$  do
7:     Assign full demands of request  $j$  to the agent 1:  $z_{1,j} \leftarrow d_j$ ;
8:   end for
9:   Output the selected set  $A$  and assignments  $z_{i,j}$  for all  $i \in A$  and  $j \in J$ ;
10: end if

```

For the capacitated case with $\beta = s_i$, different specifications of α will be studied. Prior to that, the following necessary condition can be easily observed.

Lemma 1. *For the capacitated case $\text{Feasibility}(\alpha, s_i, *)$ with any specification of α , if a feasible solution exists, then the sum of the maximum capacity for all agent must be at least as large as the total demand, i.e. $S \geq D$.*

Consider the case $\text{Feasibility}(w = 0, s_i, *)$ which relaxes the MQC constraint. We can obtain its feasible solution efficiently as follows.

Theorem 6. *$\text{Feasibility}(w = 0, s_i, *)$ can be solved in $O(nm)$ time.*

Proof. Maintain two lists. One contains all the unsatisfied requests and the other stores all the available agents. Let u denote the first unsatisfied request with remaining demand $d'_u > 0$, and p the first available agent with remaining capacity $s'_p > 0$. Accordingly, if $s'_p \geq d'_u$, we can assign all the remaining demands, d'_u , to the agent, p , decrease the remaining capacity, s'_p , and remove the request, u , from the request list; otherwise, we only assign s'_p demands of u to p , decrease the d'_u , and remove p from the agent list. The above process is continued until all the requests have been satisfied. It is easy to see that if the total agents' maximum capacity is at least as large as the total demands ($S \geq D$), the above process will stop with a feasible solution; otherwise no feasible solution exists by Lemma 1. Note that for each loop, either a request or an agent is removed and that there are n requests and m agents. Therefore, the time complexity in the loop is $O(n + m)$. Considering that initializations consume $O(nm)$ time, the total time complexity is $O(nm)$. \square

In the rest of this section, the MQC constraint is studied for feasibility cases, by starting with $\text{Feasibility}(w = \lambda, s_i, *)$, where the MQC region number w equals a fixed integer λ .

For each agent $i \in A$ and region $r \in U$, let

$$t_{i,r} = \sum_{j \in J_r} z_{i,j} \quad (6)$$

denote the total regional shipment assigned to a selected agent i among all transportation requests in the region $r \in R$. (Recall that the universal set is $U = R \cup \{0\}$, where the region 0 holds request set J_0 not constrained by MQC constraint by fixing the minimum quantity $b_0 = 0$.)

Our basic idea to solve the case $\text{Feasibility}(w = \lambda, s_i, *)$ consists of the following stages:

- Stage 1: Select all the agents, i.e. $A = \{1, 2, \dots, m\}$;
- Stage 2: For each agent $i \in A$ and region $r \in U$, generate the total regional shipment $t_{i,r}$;
- Stage 3: Based on A and $t_{i,r}$, obtain the assignment $z_{i,j}$ for $i \in A$ and $j \in J$.

As discussed before, all agents are selected in Stage 1 for the feasibility case. To see what constraints the total regional shipment $t_{i,r}$ needed to satisfy, let us examine any feasible assignment $z_{i,j}$ for $\text{Feasibility}(w = \lambda, s_i, *)$. the demand

constraint (2) implies that the total shipments to region r should be the same as its total demands for $r \in U$, i.e.

$$\sum_{i \in A} t_{i,r} = \sum_{j \in J_r} d_j. \quad (7)$$

The capacity constraint (3) implies that the total shipments to all regions by one agent i should not exceed its maximum capacity s_i for $i \in A$, i.e.

$$\sum_{r \in U} t_{i,r} \leq s_i. \quad (8)$$

And the MQC constraint (5) implies that for each agent $i \in A$ and each MQC region $r \in R$, the total regional shipment $t_{i,r}$ must satisfy:

$$t_{i,r} = 0 \text{ or } t_{i,r} \geq b_r. \quad (9)$$

Then, a necessary condition of the feasibility follows, for the case $Feasibility(w = \lambda, s_i, *)$.

Lemma 2. *If there exists a feasible assignment $z_{i,j}$ for $i \in A$ and $j \in J$ for the case $Feasibility(w = \lambda, s_i, *)$, we can have its total regional shipment $t_{i,r}$ by (6) to satisfy the constraints (7)–(9), where $i \in A$ and $r \in U$.*

Moreover, this condition is also sufficient. We show it by proving the following result, which partially solves the case $Feasibility(w = \lambda, s_i, *)$.

Lemma 3. *Algorithm 2 generates a feasible assignment $z_{i,j}$ with $i \in A$ and $j \in J$ for the case $Feasibility(w = \lambda, s_i, *)$ in $O(nm)$ time, if the given total regional shipment $t_{i,r}$ for $i \in A$ and $r \in U$ such that the constraints (7)–(9) are satisfied.*

Proof. Suppose that the total regional shipment $t_{i,r}$ satisfies constraints (7)–(9) for $i \in A$ and $r \in U$. Note that the regional request sets $J_0, J_1, \dots, J_\lambda$ forms a disjoint partition of the whole set J . We can obtain feasible assignments $z_{i,j}$ of all the agents i , for requests j in $J_0, J_1, \dots, J_\lambda$ respectively.

For each request set J_r of a region r , consider the following instance of the case $Feasibility(w = 0, s_i, *)$. The request set becomes J_r holding requests only in region r , and the demand of request j is still d_j . The agent set is A , and the maximum capacity of agent i changes to $t_{i,r}$.

In this new $Feasibility(w = 0, s_i, *)$ instance, by the constraint (7), we can apply Theorem 6 to generate feasible assignments $z_{i,j}$ for $i \in A$ and $j \in J_r$. Because the demand is unchanged, $z_{i,j}$ also satisfies the original demand constraint (2). Since $t_{i,r} \geq b_r$ by the constraint (9), the original MQC rule (5) is satisfied for region r .

Therefore, we solve those $\lambda + 1$ instances of the case $Feasibility(w = 0, s_i, *)$ for $J_0, J_1, \dots, J_\lambda$ one by one. Afterwards, assignments $z_{i,j}$ can be obtained for $i \in A$ and $j \in J$, such that both the original demand constraint (2) and MQC rules (5) are satisfied. Recall that for each region $r \in U$ the relation (6) holds

Algorithm 2 Solving $Feasibility(w = \lambda, s_i, *)$ given Total Regional Shipments

-
- 1: Input: The selected set $A = \{1, 2, \dots, m\}$, and the total regional shipment $t_{i,r}$ ($i \in A$ and $r \in U$) which satisfies constraints (7)–(9);
 - 2: Initially, set $z_{i,j} \leftarrow 0$ for $i \in I$ and $j \in J$;
 - 3: **for all** region $r \in U$ **do**
 - 4: Construct an instance of $Feasibility(w = 0, s_i, *)$, where the request set is J_r with each request $j \in J_r$ having demand d_j , and the agent set is A with each agent $i \in A$ holding maximum capacity $t_{i,r}$;
 - 5: Solve this instance by Theorem 6 to obtain a feasible assignment $z_{i,j}$ for $i \in A$ and $j \in J_r$;
 - 6: **end for**
 - 7: Output assignments $z_{i,j}$ for $i \in A$ and $j \in J$;
-

true, by (8), we know that $z_{i,j}$ satisfies its original maximum capacity constraint (3) for $i \in A$ and $j \in J$.

A formal statement of the above process is given in Algorithm 2. Note that $Feasibility(w = 0, s_i, *)$ has been solved for $|U|$ times, each consuming $O(|J_r|m)$ time for $0 \leq r \leq \lambda$. So totally, the time complexity of Algorithm 2 is $O(nm)$, by $|J_0| + |J_1| + \dots + |J_\lambda| = n$. \square

To completely solve the case $Feasibility(w = \lambda, s_i, *)$, we need to obtain the total regional shipment $t_{i,r}$ to satisfy constraints (7)–(9) for $i \in A$ and $r \in U$, before we apply the Algorithm 2. This can be achieved by a dynamic programming algorithm proposed as follows.

Let $I(p) = \{1, \dots, p\}$ denote the set of the first p agents in A ($1 \leq p \leq m$). Let $D_r = \sum_{j \in J_r} d_j$ denote the total demands for requests in region $r \in U$. To satisfy demands in the constraint (7), we trace the partial regional shipments transported by those agents $i \in I(p)$ for region r , where p is from 1 to m . In other words, for each p , let $v_r = \sum_{i \in I(p)} t_{i,r}$ denote the partial regional shipments for region r . So the states of the dynamic programming algorithm are (p, \vec{v}) , where the vector $\vec{v} = \langle v_0, v_1, \dots, v_\lambda \rangle$ and $v_r \leq D_r$ for all $r \in U$. The algorithm recursively computes values of $G(p, \vec{v})$, denoting the true value of the statement: there exists integer values of $t_{i,r}$ for $i \in I(p)$ and $r \in U$, such that

$$\sum_{i \in I(p)} t_{i,r} = v_r, \text{ for all } r \in U; \quad (10)$$

$$\sum_{r \in U} t_{i,r} \leq s_i, \text{ for all } i \in I(p); \quad (11)$$

$$t_{i,r} = 0 \text{ or } t_{i,r} \geq b_r, \text{ for all } i \in I(p) \text{ and } r \in U. \quad (12)$$

It is easy to see that there exists $t_{i,r}$ to satisfy these constraints (7)–(9) for $i \in A$ and $r \in U$, if and only if $G(m, \langle D_0, D_1, \dots, D_\lambda \rangle)$ is true. So our objective is to obtain the value of $G(m, \langle D_0, D_1, \dots, D_\lambda \rangle)$, which can be computed recursively as follows.

Initially, set $G(0, \vec{v})$ true if $\vec{v} = \langle 0, 0, \dots, 0 \rangle$; otherwise set it false.

In each iteration $p = 1, 2, \dots, m$, consider the vector \vec{v} with field $v_r \leq D_r$ for $r \in U$. To obtain the current value of $G(p, \vec{v})$, we need to enumerate all the possible values of $t_{p,r}$ for $r \in U$. Note that $t_{p,r}$ can not exceed v_r by (10).

Let Ψ be the set of vectors $\vec{t} = \langle t_{p,0}, t_{p,1}, \dots, t_{p,\lambda} \rangle$ with $t_{p,r} \leq v_r$ for $r \in U$ and satisfying (11) and (12). Accordingly by (10), we can obtain the following recursion: the value of $G(p, \vec{v})$ is true if and only if there exists a vector $\vec{t} \in \Psi$ such that the value of $G(p-1, \vec{v} - \vec{t})$ is true.

After these m iterations, we can examine the value of $G(m, \langle D_0, D_1, \dots, D_\lambda \rangle)$. If it is true, then the feasible values of $t_{i,r}$, satisfying constraints (7)–(9) for $i \in A$ and $r \in U$, can be derived based on the previous process of recurrent computations; otherwise no feasible $t_{i,r}$ exists. Whenever a feasible $t_{i,r}$ is obtained, we can solve $Feasibility(w = \lambda, s_i, *)$ by Algorithm 2. By a straightforward estimation of its time complexity, we can derive the following theorem.

Theorem 7. *$Feasibility(w = \lambda, s_i, *)$ can be solved in $O(mS^{2\lambda+2} + nm)$ time.*

So far, a pseudo-polynomial algorithm has been designed for $Feasibility(w = \lambda, s_i, *)$. By specifying $w = 2$, we have:

Corollary 1. *Finding a feasible solution for the case $Feasibility(w = 2, s_i, *)$ can be solved in $O(mS^6 + nm)$ time.*

On the other hand, the case $Feasibility(w = 2, s_i, *)$ is binary \mathcal{NP} -hard, indicating that polynomial algorithm is unlikely to exist. It can be proved by a reduction from the **Partition** problem [5]. (See [8].)

Theorem 8. *$Feasibility(w = 2, s_i, *)$ is binary \mathcal{NP} -hard.*

The complexity result above can be applied to $Feasibility(w = \lambda, s_i, *)$, by the reduction from $w = 2$ to $w = \lambda$ in the field α shown in Section 2. This derived the following corollary.

Corollary 2. *$Feasibility(w = \lambda, s_i, *)$ is binary \mathcal{NP} -hard.*

We have seen that the capacitated feasibility case of $w \geq 2$ is binary \mathcal{NP} -hard and has a pseudo-polynomial algorithm. However, when $w = 1$, an efficient polynomial algorithm exists for $Feasibility(w = 1, s_i, *)$ where $R = \{1\}$ and $U = \{0, 1\}$. Recall that the polynomial Algorithm 2 can be still applied for the case $Feasibility(w = 1, s_i, *)$ to obtain feasible assignments $z_{i,j}$ for $i \in A$ and $j \in J$, if given the total regional shipments $t_{i,r}$ for $i \in A$ and $r \in U$. So, we only need to consider how to obtain $t_{i,r}$ efficiently for this case as follows.

Suppose $S \geq D$ by Lemma 1. Assume that the agents are ordered non-decreasingly on their maximum capacities, i.e. $s_1 \geq s_2 \geq \dots \geq s_m$. Moreover, let c_1 indicate the number of agents, which can serve requests in region 1 whose capacities are at least b_1 .

If the total regional shipment $t_{i,r}$ has been obtained for $i \in A$ and $r \in U$ with satisfaction of constraints (7)–(9), it is easy to see that that the $\sum_{i=1}^c s_i \geq D_1$ must be satisfied, where $c = \min(c_1, \lfloor D_1/b_1 \rfloor)$.

To see this condition is also sufficient, we will show that if $\sum_{i=1}^c s_i \geq D_1$, we can safely choose $A_1 = \{1, \dots, c\}$ to serve the requests in region 1 and obtain $t_{i,r}$ as follows. Firstly, excluding quantity b_1 reserved for the minimum quantity of region 1, the remnant capacity of agent i can be denoted by $s'_i = s_i - b_1$ for $1 \leq i \leq c$ and the total remnant demands in region 1 is reduced to $D'_1 = D_1 - cb_1$.

By $\sum_{i \in A_1} s_i \geq D_1$, we have $\sum_{i \in A_1} s'_i \geq D'_1$. Let τ such an index that $c \geq \tau \geq 1$ and $\sum_{i=1}^{\tau} s'_i \geq D'_1 > \sum_{i=1}^{\tau-1} s'_i$. Then the first τ agents can satisfy the remnant demands D'_1 , since we can assign s'_i to each agent i where $i < \tau$, and assign the rest $D'_1 - \sum_{i=1}^{\tau-1} s'_i$ to agent τ . Furthermore, including the reserved quantity b_1 , we can satisfy all the demands D_1 by assigning agents in A_1 with $t_{i,1} = s_i$ if $i \leq \tau - 1$, and $t_{\tau,1} = D_1 - (c - \tau)b_1 - \sum_{i=1}^{\tau-1} s_i$, and $t_{i,1} = b_1$ if $\tau < i \leq c$. It is easy to verify that both the demand constraint (7) and the MQC constraints (9) are satisfied for region 1. On the other hand, for region 0, note that the total demands is $D_0 = D - D_1$ and the total remnant capacity of all agents is $S_0 = S - D_1$. By $S \geq D$, we have $S_0 \geq D_0$. Using the same way discussed above for satisfying the remnant demands D'_1 in region 1, we can obtain the regional shipment $t_{i,0}$ for $i \in A$ to satisfy constraints (7)(9) for region 0. Moreover, it is easy to see that the capacity constraint (8) for $t_{i,r}$, where $i \in A$ and $r \in U$, is satisfied by all.

As $t_{i,r}$ has been obtained for $i \in A$ and $r \in U$, we can employ Algorithm 2 to solve the feasible $z_{i,j}$ for $i \in A$ and $j \in J$. Because choosing agents with c largest capacities consumes $O(m)$ time [3], the main process in Algorithm 2 cost $O(n + m)$ time at most, except the initializations which consumes $O(nm)$ time. So, its total time complexity is $O(nm)$. This leads the following theorem.

Theorem 9. *Feasibility($w = 1, s_i, *$) can be solved in $O(nm)$ time.*

Lastly, the general capacitated feasibility case with arbitrary w can be proved to be unary \mathcal{NP} -hard by a reduction from **3-Partition** problem [5]. (See [8].)

Theorem 10. *Finding a feasible solution for the case Feasibility($w, s_i, *$) is unary \mathcal{NP} -hard.*

References

1. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows : theory, algorithms, and applications*. Prentice Hall, Englewood Cliffs, N.J., 1993.
2. B. Bozkaya, J. Zhang, and E. Erkut. A genetic algorithm for the p-median problem. In Z. Drezner and H. Hamacher, editors, *Facility Location: Applications and Theory*. Springer, Berlin, 2002.
3. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT press, Cambridge, Massachusetts, 1997.
4. G. Cornuèjols, G. L. Nemhauser, and L. A. Wolsey. The uncapacitated facility location problem. In P. Mirchandani and R. Francis, editors, *Discrete Location Theory*, pages 119–171. Wiley, New York, NY, 1990.
5. Michael R. Garey and David S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. Freeman, New York, 1983.
6. K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *Proceedings of 34th ACM Symposium on Theory of Computing (STOC)*, pages 731–740, 2002.
7. James Orlin. A faster strongly polynomial minimum cost flow algorithm. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 377–387. ACM Press, 1988.
8. Zhou Xu. On the selection and assignment problem with minimum quantity commitments. Master's thesis, National University of Singapore, Sept. 2003. <http://www.comp.nus.edu.sg/~judge/doc.html>.

Approximation Algorithms for Multicommodity Flow and Normalized Cut Problems: Implementations and Experimental Study^{*}

Ying Du^{1,**}, Danny Z. Chen^{1,**}, and Xiaodong Wu^{2,***}

¹ Dept. of Computer Science and Engineering,
University of Notre Dame, Notre Dame, IN 46556, USA
{ydu1,dchen}@cse.nd.edu

² Dept. of Computer Science, University of Texas – Pan American, Edinburg, TX
78539, USA
xwu@cs.panam.edu

Abstract. In this paper, we present (1) our implementation of a fast approximation multicommodity flow algorithm, and (2) our implementation of the first provably good approximation algorithm for the minimum normalized cut problem. (2) serves as an important application of (1). Our experimental results show that the output quality of our approach compares favorably against some previous approximation multicommodity flow implementation and eigenvalue/eigenvector based normalized cut implementation. We also show comparisons on the execution times and analyze the underlying reasons.

1 Introduction

The multicommodity flow problem [1] is concerned with computing the largest rate at which flow commodities can be routed from multiple sources to multiple sinks in a graph without violating any given edge capacity constraints. This problem is essential to solving many optimization problems, and finds substantial applications in the area of approximation algorithms.

The goals of our work are to obtain a fast implementation of a good quality approximation algorithm for the multicommodity flow problem, and to use this implementation as a main tool to study some important applications. For example, the maximum concurrent flow problem (MCFP) [19] that we study can be used to solve the pairwise data clustering problem [25] based on a novel criterion called *normalized cut* [24].

^{*} This research was supported in part by the 21st Century Research and Technology Fund from the State of Indiana.

^{**} This research was supported in part by the National Science Foundation under Grant CCR-9988468.

^{***} This research was supported in part by a grant from the University of Texas – Pan American Faculty Council and a grant from the Computing and Information Technology Center at the University of Texas – Pan American, Edinburg, Texas, USA.

The normalized cut criterion measures both the total dissimilarity among different components of a partition as well as the total similarity within the same components. Shi and Malik [24] showed that the minimum normalized cut criterion is capable of delivering impressive image segmentation results. Since the minimum normalized cut problem is NP-complete [24], they gave a heuristic algorithm for solving this problem by computing the eigenvalues and eigenvectors of the incident Laplacian matrix of the corresponding graph. However, no provable quality is known for their approach.

Wu *et al.* [25] proposed the first provably good polynomial time combinatorial approximation algorithm for the minimum normalized cut problem, by modeling it approximately as a linear program (LP) dual of the maximum concurrent flow problem [19] and applying the sphere-growing technique [8, 11]. For an undirected weighted graph of n vertices and m edges and any fixed value $\epsilon > 0$, they presented a $(4(1+\epsilon) + o(1)) \ln(n)$ -approximation algorithm in $\tilde{O}(\frac{1}{\epsilon^2} m^2)$ time for the minimum normalized cut problem, where $\tilde{O}(f(m))$ denotes $O(f(m) \log^{O(1)} m)$. The sphere-growing procedure essentially runs Dijkstra's shortest path algorithm, in $O(n \log n + m)$ time [7]. The approach in [25] also improved the approximation ratio of the algorithm for computing the sparsest cut [11] by a factor of nearly 4. Note that the time for solving the linear program (LP) dual of the concurrent flow problem dominates the overall running time of the algorithm [25]. Previous experimental results have suggested that concurrent flow based approximation techniques can yield much faster running time.

Given a graph $G = (V, E)$ with non-negative edge capacities $u(e)$ and $k \geq 1$ commodities, such that s_i and t_i are the source and sink for commodity i associated with a demand $d(i)$, the *maximum concurrent flow* problem (MCFP) [19] seeks the largest possible rate λ such that there is a multicommodity flow that routes $\lambda \cdot d(i)$ units of commodity i simultaneously in G for all i . Note that in Wu *et al.*'s approximation minimum normalized cut algorithm, it needs to solve a *product multicommodity flow problem* (PMFP), which is a special MCFP with the demand of flow between every pair of vertices u and v being the product of u and v 's weights. The MCFP is the dual of the LP relaxation of the sparsest cut problem [11], and can be solved optimally in polynomial time using LP techniques. However, in practice, the LP based approach may not be practically efficient for large size normalized cut problems.

For many applications, it is more important to solve the MCFP approximately in a fast manner than to solve it optimally. Developing fast approximation algorithms for the MCFP to produce solutions that are provably close to the optimal one has attracted a lot of attention. A series of papers provides *fully polynomial time approximation scheme* (FPTAS) for the problem, based on Lagrangian relaxation and LP decomposition techniques (e.g., [12, 16–18, 21, 22]). Building on the framework of [9, 10], Karakostas [15] gave a faster FPTAS for the MCFP. This algorithm (to be described in Section 2.3) eliminates the running time dependence on k , and runs in $\tilde{O}(\epsilon^{-2} m^2)$ time, improving the previous best $\tilde{O}(\epsilon^{-2} m(m+k))$ time MCFP algorithm [9]. To our knowledge, Karakostas' algorithm [15] has the so far best running time.

In this paper, we report the following work.

- Our implementation of the approximation algorithm in [15] for the MCFP.
- Our implementation of the approximation minimum normalized cut algorithm in [25]. This uses our implementation of [15] as the main tool, and serves as an important application of our implementation of the MCFP.
- A study of the experimental behaviors of these two algorithms [15, 25]. Our experimental data show that the practical performance of the algorithms is much better than their theoretical time bounds.

Implementing Karakostas' MCFP algorithm is a nontrivial effort since the algorithm is quite sophisticated. To improve the running time in practice, we must modify certain details of the algorithm [15], add and fine-tune some scale parameters to control the computations, and apply a number of strategies to reduce numerical errors. We describe our changes and the motivations behind them in Sections 3. We show and analyze our experimental results in Section 4.

2 Preliminaries

2.1 Minimum Normalized Cuts

Given an undirected graph $G = (V, E)$, with each edge $e \in E$ having a weight $w(e) \geq 0$, a *cut* C of G is a subset of edges whose removal partitions V into two disjoint complementary subsets A and \bar{A} . The *minimum normalized cut* is a cut C whose *normalized cost*

$$\alpha(A) = \frac{\text{cut}(A, \bar{A})}{\text{assoc}(A, V)} + \frac{\text{cut}(A, \bar{A})}{\text{assoc}(\bar{A}, V)} \quad (1)$$

is minimized, where $\text{cut}(A, \bar{A}) = \sum_{e \in C} w(e)$ is the sum of edge weights of the cut C , and $\text{assoc}(S, V) = \sum_{(u,v) \in E, u \in S, v \in V} w(u, v)$ for a subset S of V is the total weight of edges between S and all vertices in G . Papadimitriou [24, 27] has shown that computing a minimum normalized cut in G is an NP-complete problem.

2.2 Wu *et al.*'s Combinatorial Algorithm

Wu *et al.* [25] studied the minimum normalized cut problem on an undirected graph $G = (V, E)$, with each edge e having a weight $w(e) \geq 0$. The weight $w(u)$ of each vertex u in G is defined as $\frac{1}{2} \sum_{e=(u,v) \in E} w(e)$. For a subset $A \subseteq V$, let $w(A)$ denote the total weight of the vertices in A , and W denote the total weight of all edges in E . Then W is also equal to $\sum_{v \in V} w(v)$. The algorithm [25] first introduces a direct fractional relaxation of the minimum normalized cut problem as an LP dual of the product multicommodity flow problem [19]. This linear program assigns a non-negative edge length $l(e)$ to each edge $e \in E$, which will be used by the sphere-growing procedure (to be discussed later). To apply the linear fractional relaxation, a commodity is associated with every pair of vertices u and v in G , whose demand is set as $w(u) \cdot w(v)$.

We need to review some notation. For a subgraph $G' = (V', E')$ of G , let $N(v, r) = \{u \in V' \mid \text{dist}_{G'}(u, v) < r\}$ denote a *sphere* of radius r in G' that grows from a vertex $v \in V'$, where $\text{dist}_{G'}(u, v)$ is the u -to- v shortest path length in G' based on the edge length function $l(\cdot)$. Let $E(v, r)$ denote the set of edges whose end vertices are both in $N(v, r)$, $C(v, r)$ denote the induced cut of $N(v, r)$ (i.e., $C(v, r)$ is the set of edges cutting $N(v, r)$ from $V - N(v, r)$), and $w(C(v, r))$ denote $\sum_{e \in C(v, r)} w(e)$. The *volume* of a sphere $N(v, r)$, denoted by $\text{vol}(v, r)$, defines the *credits* that are associated with the sphere in order to pay for the cost of the induced cut (see [25] for the precise definition).

The basic idea of the sphere-growing process is to run Dijkstra's single-source shortest path algorithm in G' from an arbitrary vertex v to produce a sphere $N(v, r)$, such that the ratio R of $w(C(v, r))$ over $\text{vol}(v, r)$ is logarithmic in terms of n . The procedure starts at an arbitrary vertex $v \in V'$, puts v into a sphere denoted by S , and finds a vertex x such that $\text{dist}_{G'}(v, x)$ is the smallest of $\text{dist}_{G'}(v, v')$ among all $v' \in V' - S$. It then checks whether $w(C(v, r)) > \sigma \cdot \text{vol}(v, r)$. If yes, then x is put into the sphere S , and $r = \text{dist}_{G'}(v, x)$. The procedure repeatedly puts new vertices into S , until either the total weight $w(C(v, r))$ of the cut is no bigger than the criterion given above or no more vertex is left in $V' - S$, i.e., the termination condition of the sphere-growing procedure is $R \leq \sigma$ or $V' = S$. The value of σ is a key in determining the upper bound of the sphere radius r thus computed, and is chosen as $\frac{W^2}{\sqrt{2}} \sqrt{\ln(n \ln n + 1)} \cdot (\sqrt{2} \sqrt{\ln(n \ln n + 1)} + 1)$ in [25], which yields a better approximation ratio than the previous work.

Wu *et al.*'s algorithm uses this sphere-growing procedure as a basic function, and generates a set of disjoint spheres in G . In all spheres thus produced, it has been proved [25] that either one can find a sphere which gives the cut that yields a $(4 + o(1)) \ln n$ -approximate result, or can choose a vertex of G from which another sphere can be grown to generate the desired normalized cut (see [25] for more details).

2.3 Maximum Concurrent Flows

We now summarize Karakostas' approximation algorithm [15] for solving the MCFP. As mentioned before, the MCFP can be expressed as a linear program. Let $D(l)$ denote the dual objective function $\sum_e u(e) \cdot l(e)$, $\text{dist}_i(l)$ denote the length of the shortest path from s_i to t_i in G based on the length function l , and $\alpha(l) = \sum_i d(i) \cdot \text{dist}_i(l)$. It has been observed [10] that minimizing $D(l)$ under the dual constraints can also be viewed as computing the lengths $l(e)$ for the edges e of G , such that $D(l)/\alpha(l)$ is minimized. Let β denote the minimum value of $D(l)/\alpha(l)$.

When $\beta \geq 1$, Karakostas [15] gave an $\tilde{O}(\epsilon^{-2} m^2)$ time approximation algorithm that improves on [10]. Initially, for all edges e , $l(e) = \delta/u(e)$, where δ is a fixed value that is set to $(1 + k\epsilon)^{-\frac{1-\epsilon}{\epsilon}} \cdot (\frac{1-\epsilon}{m})^{\frac{1}{\epsilon}}$. The algorithm proceeds in phases. In each phase, there are $|S|$ iterations, where S is the set of source vertices. In iteration j , consider all commodities c_1, c_2, \dots, c_r that share the same source s_j . The objective in this iteration is: For all demands $d(c_q)$, $q = 1, 2, \dots, r$, route

$d(c_q)$ units of flow from s_j to the corresponding sink of commodity c_q . This is done in a series of steps. In each step, the shortest paths from s_j to the sink of each commodity c_q is computed based on the current length function. Let c be the minimum capacity of the edges on the shortest paths thus computed for all commodities c_q . Then for each commodity c_q , an amount of flow that is equal to the smallest value of c and the remaining demand of c_q is sent from s_j along the shortest path for c_q . The amount of routed flow is deducted from the demand of c_q , and the edge lengths on these shortest paths are updated according to a length function $l(e) = l(e)(1 + \epsilon \cdot \frac{\sum_{c_q: e \in P_{c_q}} f_{c_q}}{u(e)})$, where f_{c_q} is the amount of flow routed for commodity c_q and P_{c_q} is the shortest path from s_j to the sink of commodity c_q . The entire procedure terminates when the dual objective function $D(l) = \sum_e u(e) \cdot l(e) \geq 1$.

When $\beta < 1$, it has been shown in [10] that one can scale the ratio of capacities over demands, and use some binary search type methods to ensure $1 \leq \beta \leq 2$.

3 Implementations

We implemented the algorithms using programming languages C/C++ on a Sun Blade 1000 Workstation with a 600 MHz UltraSPARC III Processor and 512 Megabytes of memory. In this section, we describe some important implementation details and our handling of the issues arising in our implementations.

As mentioned before, Wu *et al.*'s algorithm works on undirected graphs, while Karakostas' algorithm is designed for directed graphs. To make our implementations compatible with both of these two algorithms, for each edge in the original undirected graph, we add 2 more vertices and 4 more edges to form an "equivalent" directed subgraph [1]. By carefully assigning the edge lengths and weights, we ensure the correctness of the flow results.

The maximum concurrent flow algorithm in [15] proceeds in phases; each phase consists of iterations and each iteration performs some steps. When the whole algorithm terminates, the flow values have been scaled down to a solution that is very close to the optimal one without violating any edge capacities (i.e., a feasible and good approximation quality solution). Our experiments have shown that on some problem instances for which the feasible, exact optimal solutions can be easily found (i.e., all commodity demands can be routed simultaneously without violating any edge capacities), it is really not necessary to waste time to route the flow using multiple phases. In this case, we can actually find a feasible, exact optimal solution in one phase, reducing the running time dramatically.

To achieve this goal and for the purpose of speeding up the program for all cases, we use an extra copy of the edge capacities of the graph. This copy of the edge capacities is modified as the algorithm proceeds while the original copy will not be changed. At the beginning of each iteration, the edge capacities in the extra copy are reset to the edge capacities of the original graph, and they are updated after routing the flow in each step. We also use a flag to indicate whether there is an edge whose flow already exceeds its capacity. As discussed

in Section 2, after the last step of iteration j , all demands of the commodities with the same source s_j have been routed. Thus, after all iterations, if the flag has not been set, then a feasible and exact optimal flow solution has been found, and we can stop the algorithm. In this way, a feasible, exact optimal solution can be obtained after only 1 phase of $|S|$ iterations, where S is the number of different sources.

Correspondingly, the edge lengths along the computed shortest paths are updated according to a function $l(e) = l(e)(1 + \epsilon \cdot \frac{\sum_{c_q: e \in P_{c_q}} \text{flow}(e)}{u_{\text{copy}}(e)})$, where $\text{flow}(e)$ denotes the total flow routing on edge e and $u_{\text{copy}}(e)$ denotes the remaining weight of e in the extra edge capacity copy. If $u_{\text{copy}}(e)$ of an edge e becomes zero, namely, the flow routing on this edge has used up all its capacity, then we use another length function $l(e) = 2000 \cdot l(e)(1 + \epsilon \cdot \frac{\sum_{c_q: e \in P_{c_q}} \text{flow}(e)}{u(e)})$. The purposes of using different length functions are to assign larger lengths (costs) to edges with less capacities left, and to encourage routing flows along other less congested edges in the subsequent steps. The scale-up factor reduces the possibility of routing flows on edges which are already “full” when some “unfull” paths still exist.

Numerical errors occur frequently in our implementations. To deal with these problems, several strategies have been applied in our implementations to achieve higher reliability. For example, we use “double” data type, avoid/delay “inaccurate” computations such as square root, apply some static error estimates schemes, scale up small values if the results will not be affected, and etc..

4 Experimental Results

We have run our programs on a variety of problems. The experiments were performed on the same Sun Blade 1000 Workstation. All graphs shown below are generated by a random graph generator written by ourselves in C++.

4.1 Dependence on ϵ

The maximum concurrent flow algorithm in [15] takes $\tilde{O}(\epsilon^{-2}m^2)$ time, where $\tilde{O}(f(m))$ denotes $O(f(m) \log^{O(1)} m)$. Thus, the theoretical bound shows an inverse polynomial dependence of the execution time on the parameter ϵ for the approximation error.

We ran our MCFP program on various random graphs with randomly placed commodities. Let n , m , and k denote the numbers of vertices, edges, and commodities of an input undirected graph, respectively. For example, using a random graph with $n = 40$ vertices, $m = 72$ edges, and $k = 20$ commodities, we graphed the ϵ values versus the total execution time (in seconds) in Figure 1. The upper curve represents the theoretical time bound that we interpolate using the *least squares* method. The lower curve shows the actual execution time for different ϵ . Each point on the lower curve represents an average execution time of 10 runs. Figure 1 shows that as the ϵ value decreases, the total execution time tends to

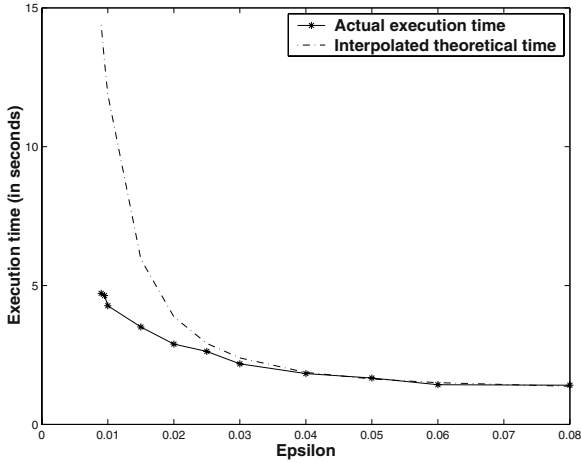


Fig. 1. Illustrating the dependence of the execution time on ϵ

increase polynomially; the overall pattern of the lower curve seems to indicate an inverse polynomial dependence of the execution time on ϵ . The figure also shows that as ϵ decreases, the actual execution time increases in a much slower manner than the predicted theoretical time bound. This suggests that as ϵ decreases, the algorithm on average tends to run much faster in practice than the worst-case theoretical time bounds.

4.2 Running Time vs. the Number of Commodities or Number of Edges

The MCFP algorithm in [15] shows that by grouping the commodities with the same source in the process of each iteration, the dependence of the total running time on the number of commodities can be virtually eliminated.

We tested this hypothesis on a set of maximum concurrent flow problems with the same numbers of vertices and edges, but different numbers of commodities. The results show that the total execution time is affected slightly by the number of commodities in certain range, due to the change of the β value. When the number of commodities increases, the β value gets smaller, thus the total execution time actually decreases. When β drops below 1, the scaling procedure ensures $1 \leq \beta \leq 2$, hence after that, the execution time becomes pretty stable as the number of commodities changes.

The experimental results also show that with the same commodity number, the total execution time increases polynomially as the number of edges increases.

4.3 Comparison with Leong *et al.*'s Implementation

Based on our experience, Leong *et al.*'s implementation is one of the best codes that we know for approximately solving the MCFP. Thus, we compare our imple-

Table 1. Comparison of our execution time (in seconds) with Leong *et al.*'s (with $\epsilon = 0.01$)

Problem size (n, m, k)	Our work	Leong <i>et al.</i> 's
(40,86,60)	2.47	7.51
(50,100,60)	4.22	4.97
(55,128,150)	7.33	169.60
(100,202,120)	17.40	25.96
(180,413,250)	85.25	114.75
(250,587,300)	204.36	87.41
PMFP		
(20,47,380)	2.76	344.92
(40,222,1560)	2.99	5.32
(50,305,2450)	10.15	15.74
(100,661,9900)	35.19	102.63
(200,706,39800)	136.96	205.41
(300,1064,89700)	500.40	146.38

Table 2. Comparison of our output quality (in percentage) with Leong *et al.*'s (with $\epsilon = 0.01$)

Problem size (n, m, k)	Our work	Leong <i>et al.</i> 's
(40,86,60)	39.37	23.11
(50,100,60)	17.30	33.33
(55,128,150)	37.79	21.36
(100,202,120)	25.60	16.67
(180,413,250)	17.60	16.95
(250,587,300)	19.36	16.67
PMFP	%	%
(20,47,380)	45.12	8.07
(40,222,1560)	28.26	6.67
(50,305,2450)	26.18	7.16
(100,661,9900)	6.59	3.35
(200,706,39800)	1.61	0.15
(300,1064,89700)	0.75	0.03

mentation with theirs on both the general MCFP and PMFP. Some comparison results on the execution time are shown in Table 1, and some comparison results on the output quality are shown in Table 2.

The execution times in Table 1 represent the average times of 10 runs per entry. We noticed that our implementation does not really have much advantage on the execution time over Leong *et al.*'s. In the 231 problem instances we tested, for 61.9% of the test cases, our algorithm performs faster than Leong *et al.*'s algorithm; for 92.61% of the test cases, our algorithm produces better quality results (higher percentages of the commodity demands are routed).

A possible reason that our implementation does not have obvious advantages on the execution time may be that Leong *et al.*'s code uses very straightforward and efficient data structures such as arrays, and it deals with only integer lengths, demands, and capacities. Our code is designed for more general settings and is easier to expand to accommodate additional requirements; these advantages may pay a price of incurring more overhead in the execution of our code. Note that for problem instances with larger graph sizes, Leong *et al.*'s implementation appears to have a better chance to outperform our MCFP implementation in terms of the execution time. A reason to explain this phenomenon is that their implementation uses the following heuristic: If the flow result does not improve after 2000 consecutive iterations, then their program will think that there is not much chance to converge and thus stop the execution. In such situations, their program may run faster, but according to Table 2, our approach gives flow results with a much better approximation quality, especially for PMFP instances.

4.4 Performance of Our Normalized Cut Implementation

Wu *et al.* [25] have shown that by applying the fast approximation MCFP algorithm [15], an approximate minimum normalized cut can be computed efficiently. Our experiments verified this hypothesis. For most problems we tested, the time for the PMFP computation occupies more than 90% of the total execution time

of the algorithm [25]. The PMFP computation clearly dominates the total running time for computing an approximate minimum normalized cut.

We also compare the execution times and the output approximate normalized costs between our approach and Shi and Malik's heuristic algorithm [24]. Our implementation produces a much better graph partition (smaller normalized cost) comparing to their eigenvalue/eigenvector based normalized cut approach. However, our execution time increases faster as the graph size gets larger.

4.5 Some Limitations

During the implementations and experiments, we did notice some limitations of our approaches. These limitations are the potential areas where our approaches could benefit from future research.

First, as described in Section 3, to combine the normalized cut algorithm [25] with the MCFP algorithm [15], we convert the input undirected graph to a directed graph for the MCFP computation, and then convert the directed graph back to the undirected graph for the sphere-growing process. This conversion gives rise to a larger size graph, and the MCFP computation may waste time on some "meaningless" edges.

Another difficulty is that the key parameter δ set in the approximation MCFP algorithm [15] is usually too small to be implemented efficiently. This affects both the correctness and running time of the programs. Although we have used some strategies and heuristics to deal with this problem, we have not yet found an optimal way to handle it.

Acknowledgements

We are very grateful to Dr. Clifford Stein for providing us with their maximum concurrent flow codes. We also like to thank Dr. Stein and Dr. George Karakostas for helpful discussions on the maximum concurrent flow problems.

References

1. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, 1993.
2. A. Ben-Dor and Z. Yakhini, Clustering Gene Expression Patterns, *Proc. of 3rd ACM RECOMB*, 1999, pp. 33-42.
3. D. Bienstock, talk at Oberwolfach, Germany, 1999.
4. J. Carballido-Gamio, S.J. Belongie, and S. Majumdar, Normalized Cuts in 3-D for Spinal MRI Segmentation, *IEEE Transactions on Medical Imaging*, 2003, to appear.
5. J. Chen, Z. Bai, B. Hamann, and T.J. Ligocki, A Normalized-Cut Algorithm for Hierarchical Vector Field Data Segmentation, *Proc. of Visualization and Data Analysis*, Santa Clara, CA, 2003.
6. D.Z. Chen and X. Wu, Data Clustering Based Segmentation Algorithms of Bone CT Images, *Proc. of 9th Annual Symp. of Computational Methods in Orthopaedic Biomechanics*, San Francisco, 2001, p. 19.

7. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd edition, McGraw-Hill, 2001.
8. G. Even, J. Naor, S. Rao, and B. Schieber, Fast Approximate Graph Partitioning Algorithms, *SIAM J. Computing*, 28(1999), 2187-2214.
9. L. Fleischer, Approximation Fractional Multicommodity Flow Independent of the Number of Commodities, *SIAM J. Discrete Math.*, 13(2)(2000), 145-153.
10. N. Garg and J. Könemann, Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems, *Proc. of 39th IEEE Foundation of Computer Science*, pp. 300-309, 1998.
11. N. Garg, V.V. Vazirani, and M. Yannakakis, Approximate Max-Flow Min-(Multi)Cut Theorems and Their Applications, *SIAM J. Computing*, 25(1996), 235-251.
12. M.D. Grigoriadis and L.G. Khachiyan, Approximate Minimum-Cost Multicommodity Flows, *Mathematical Programming*, 75(1996), 477-482.
13. S. Guattery and G. Miller, On the Performance of Spectral Graph Partitioning Methods, *Proc. of 6th ACM-SIAM SODA*, pp. 233-242, 1995.
14. T. Hofmann and J. Buhmann, Pairwise Data Clustering by Deterministic Annealing, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(1997), 1-14.
15. G. Karakostas, Faster Approximation Schemes for Fractional Multicommodity Flow Problems, *Proc. of 13th Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 166-173, 2002.
16. D. Karger and S. Plotkin, Adding Multiple Cost Constraints to Combinatorial Optimization Problems, with Applications to Multicommodity Flows, *Proc. of 27th ACM STOC*, pp. 18-25, 1995.
17. P. Klein, S. Plotkin, C. Stein, and É. Tardos, Faster Approximation Algorithms for the Unit Capacity Concurrent Flow Problem with Applications to Routing and Finding Sparse Cuts, *SIAM J. Computing*, 23(1994), 466-487.
18. T. Leighton, F. Makedon, S. Plotkin, C. Stein, É. Tardos, and S. Tragoudas, Fast Approximation Algorithms for Multicommodity Flow Problems, *J. of Computer and System Sciences*, 50(1995), 228-243.
19. T. Leighton and S. Rao, Multicommodity Max-Flow Min-Cut Theorems and Their Use in Designing Approximation Algorithms, *J. of the ACM*, 46(1999), 787-832.
20. T. Leong, P. Shor, and C. Stein, Implementation of a Combinatorial Multicommodity Flow Algorithm, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science: The 1st DIMACS Implementation Challenge: Network Flows and Matchings*, D. Johnson and C. McGoeck (eds.), 1993.
21. S. Plotkin, D. Shmoys, and É. Tardos, Fast Approximation Algorithms for Fractional Packing and Covering Problems, *Mathematics of Operations Research*, 20(1995), 257-301.
22. T. Radzik, Fast Deterministic Approximation for the Multicommodity Flow Problem, *Proc. of 6th ACM-SIAM SODA*, pp. 486-492, 1995.
23. G.V. Shenoy, *Linear Programming Methods and Applications*, Wiley Eastern Limited, 1989.
24. J. Shi and J. Malik, Normalized Cuts and Image Segmentation, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(8) (2000), 888-905.
25. X. Wu, D.Z. Chen, J.J. Mason, and S.R. Schmid, Pairwise Data Clustering and Applications, *Proc. of 9th Annual International Computing and Combinatorics Conference*, pp. 455-466, 2003.
26. N. Young, Randomized Routing without Solving the Linear Program, *Proc. of 6th ACM-SIAM SODA*, pp. 170-178, 1995.
27. C.H. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994.

Transshipment Through Crossdocks with Inventory and Time Windows

Andrew Lim^{1,2}, Zhaowei Miao¹, Brian Rodrigues³, and Zhou Xu¹

¹ Department of Industrial Engineering and Engineering Management,
Hong University of Science and Technology, Clearwater Bay, Hong Kong

² The Logistics and Supply Chain Institute,

Hong University of Science and Technology, Clearwater Bay, Hong Kong

³ School of Business, Singapore Management University, 469 Bukit Timah Road,
Singapore 259756

Abstract. The supply chain between manufacturers and retailers always includes transshipments through a network of locations. A major challenge in making demand meet supply has been to coordinate transshipment activities across the chain aimed at reducing costs and increasing service levels in the face of a range of factors, including demand fluctuations, short lead times, warehouse limitations and transportation and inventory costs. The success in implementing push-pull strategies, when firms change from one strategy to another in managing the chain and where time lines are crucial, is dependent on adaptive transshipment scheduling. Yet again, in transshipment through crossdocks, where just-in-time objectives prevail, precise scheduling between suppliers, crossdocks and customers is required to avoid inventory backups or delays. This paper studies transshipment with supplier and customer time windows where flow is constrained by transportation schedules and warehouse capacities with the objective to minimize costs, including inventory costs. Transportation is provided through flexible schedules and lot-sizing is dealt with in the models through multiple shipments. We propose polynomial time algorithms or show the complexity of problems studied.

1 Introduction

Making supply meet demand (Fisher et al. 1994) has been a challenge for many industries. In the face of increasing numbers of products, decreased product life cycles, inaccuracy of demand forecasts and short lead times, strategies to achieve this are a high priority for many firms. This paper studies one such a strategy, that of transshipments.

Transshipment is concerned with how much to ship between which locations on which routes and at what times. In classical models, where transshipment is studied in the context of network flow (e.g. Aronson 1989, Hoppe and Tardos 1995), there has been no work which includes inventory and time window constraints present in business models. One such model where transshipment becomes an important factor is in crossdocking which has become synonymous

with rapid consolidation and processing. Napolitano (2000) has described various crossdocking operations which include manufacturing, transportation, distributor, transportation, retail and opportunistic crossdocking, all of which have the common feature of consolidation and short cycle times made possible by known pickup and delivery times. Crossdocking networks are, in fact, difficult to manage and require significant start-up investment where crossdocking strategies are effective only for large systems (Simchi-Levi 2003). Here, we have sufficient volumes of goods which are picked up and delivered at crossdocks by vehicles with the objective of meeting demand from supply while incurring minimum or no inventory holding costs. In view of the inverse relationship between inventory and transportation costs, where larger lot sizes can reduce delivery frequency and transportation costs but increase inventory costs, transshipment from suppliers to customers through crossdocks can be managed to exploit this relationship and reduce costs only if shippers are able to operate within transportation schedules and time window constraints at all points of the supply and demand chain.

In this work, we study general transshipment networks where vehicles are available at fixed times or always available depending on transportation vendors, and where both shipping and delivery can be only be effected within time windows at supply and delivery locations. To cater to manufacturer or warehouse constraints, supplier time windows are included. This will allow, for example, the shipper flexibility in planning for best shipout times in line with his production and operating schedules. Flexible supplier shipout times can benefit attempts at integrating the "back ends" of supply chains (Simchi-Levi et al 2003) better into the chain. We include inventory holding costs at transshipment points or crossdocks as these are burdens that can arise in any crossdocking strategy. Techniques using network flow are developed which are easily applied, yet general enough to be useful in a range of problems. Where polynomial time algorithms are not possible, we provide complexity information for the problem.

This paper is organized as follows. In Section 2, we give a detailed formulation of the basic model as an LP and in Section 3 we derive solutions to the models and discuss time complexity of the models. In Section 4, we show that techniques can be used when penalties are included. Finally, we provide a short summary of this work in Section 5.

2 The Basic Model

We represent the basic problem by a bipartite network constructed as follows: Let $S \equiv \{1, \dots, n\}$ be a set of supply nodes where, for each $i \in S$, s_i units of cargo is available and can be shipped (released) in the time window $[b_i^r, e_i^r]$, $D \equiv \{1, \dots, m\}$ be a set of demand nodes each requiring d_k units of cargo which can be delivered (accepted) in the time window $[b_k^a, e_k^a]$, for $k \in D$, and $T \equiv \{1, \dots, l\}$ be transshipment centers (TCs), each with inventory capacity c_j and holding cost h_j per unit per time, for $j \in T$. We take S_1 to consist of all scheduled routes between points in S and T , i.e., available routes serviced by transport providers, where each route has scheduled departure (begin) and arrival (end)

times, unit transportation costs, and is represented in the network by a directed arc from a supply node to a TC. Similarly, we take S_2 to be the set of routes between T and D and construct directed arcs from TCs to demand nodes. Here, we assume w.l.o.g. that total supply and demand are equal since the imbalanced problem can be easily transformed using dummy-nodes. We can now introduce the following basic notation:

S : set of supply nodes

T : set of TC nodes

D : set of demand nodes

s_i : total supply at i

$[b_i^r, e_i^r]$: shipping time window at supply node i

V_{ij} : set of arcs from supply node i to TC j in S_1 , i.e., $V_{ij} = \{(t_b, t_e) : (t_b, t_e) \in S_1\}$, where t_b and t_e are the begin time and the end time, respectively, of an arc from i to j in S_1 represented by (t_b, t_e)

Γ_j : set of time points cargo arrives at TC j , i.e., $\Gamma_j = \{t_e : (t_b, t_e) \in V_{ij}, i = 1, 2, \dots, n\}$

$c_{ij, t_b t_e}$: unit shipping cost from supply node i to TC j on arc (t_b, t_e)

$k_{ij, t_b t_e}$: shipping capacity from supply node i to TC j on arc (t_b, t_e)

d_i : total demand at i

$[b_i^a, e_i^a]$: delivery time window at demand node i

V'_{jk} : set of arcs from TC j to demand point k in S_2 , i.e. $V'_{jk} = \{(t_b, t_e) : (t_b, t_e) \in S_2\}$, where t_b and t_e are the begin time and the end time, respectively, of an arc from TC j to k in S_2 .

Γ'_j : set of time points when cargo leaves TC j , i.e. $\Gamma'_j = \{t_b : (t_b, t_e) \in V'_{jk}, k = 1, 2, \dots, m\}$

$c'_{ij, t_b t_e}$: unit shipping cost from TC i to demand j on arc (t_b, t_e)

$k'_{ij, t_b t_e}$: shipping capacity from TC i to demand point j on arc (t_b, t_e)

h_i : unit inventory holding cost per time in TC i

c_i : inventory capacity of TC i

$x'_{ij, t_b t_e}$: no. of units shipped from supply i to TC j through $(t_b, t_e) \in S_1$

$x_{jk, t_b t_e}$: no. of units shipped from TC j to demand k through $(t_b, t_e) \in S_2$

y_{jt} : inventory of TC j at time t , where $t \in L_j = \Gamma_j \cup \Gamma'_j$

Here, we can sort $t \in L_j$ in increasing order, i.e., $t_1 < t_2 < \dots < t_i < t_{i+1} < \dots < t_{|L_j|}$, and take $\Delta t_i = t_{i+1} - t_i, i = 1, 2, \dots, |L_j| - 1$. Our purpose is to find the flow in the network which minimizes transportation and inventory holding costs in the objective:

$$\sum_{i \in S} \sum_{j \in T} \sum_{(t_b, t_e) \in V_{ij}} c_{ij, t_b t_e} x_{ij, t_b t_e} + \sum_{j \in T} \sum_{k \in D} \sum_{(t_b, t_e) \in V'_{jk}} c'_{jk, t_b t_e} x'_{jk, t_b t_e} + \sum_{j \in T} \sum_{t \in L_j} \Delta t_j y_{jt} \quad (1)$$

subject to the following supplier capacity (2), customer demand (3), inventory (4,5), transport capacity (6,7), inventory capacity (8) and non-negativity constraints:

$$\sum_{j \in T} \sum_{(t_b, t_e) \in V_{ij}, b_i \leq t_b \leq e_i} x_{ij, t_b t_e} = s_i \quad \text{for all } i \in S \quad (2)$$

$$\sum_{j \in T} \sum_{(t_b, t_e) \in V'_{jk}, b_k \leq t_b \leq e_k} x'_{jk, t_b t_e} = d_k \quad \text{for all } k \in D \quad (3)$$

$$y_{jt_1} = 0, \quad y_{jt_{|L_j|}} = 0 \quad (4)$$

$$y_{jt_p} = y_{jt_{p-1}} + \sum_{i \in T} \sum_{(t_b, t_p) \in V_{ij}} x_{ij, t_b t_p} - \sum_{k \in D} \sum_{(t_p, t_e) \in V'_{jk}} x'_{jk, t_p t_e} \quad (5)$$

$$\text{for all } j \in T \text{ and } p = 2, 3, \dots, |L_j| - 1$$

$$x_{ij, t_b t_e} \leq k_{ij, t_b t_e} \quad \text{for all } i \in S, \quad j \in T, \quad (t_b, t_e) \in V_{ij} \quad (6)$$

$$x'_{jk, t_b t_e} \leq k'_{jk, t_b t_e} \quad \text{for all } j \in T, \quad k \in D, \quad (t_b, t_e) \in V'_{jk} \quad (7)$$

$$y_{jt_p} \leq c_j \quad \text{for all } j \in T, \quad p = 2, 3, \dots, |L_j| - 1 \quad (8)$$

$x_{ij, t_b t_e}$, $x'_{jk, t_b t_e}$ and y_{jt} are non-negative integers for all $i \in S$, $j \in T$, $k \in D$, $(t_b, t_e) \in V_{ij}$, $(t_b, t_e) \in V'_{jk}$, $t \in L_j$

The inventory constraints (4,5) ensure that initial and final inventory levels are zero, and the inventory level at time t_p is the sum of the inventory level at time t_{p-1} and the total inbound at time t_p minus the total outbound at time t_p . In the capacity constraint (6) requires that at each TC, the inventory level at any time should be no greater than the capacity of the TC.

3 Fixed and Flexible Schedules

If the departure and arrival times in the transport schedule are fixed, we call the problem a *fixed schedule* problem. A schedule is a fixed schedule once begin times are fixed. It is common, however, that suppliers require the flexibility to ship at any time within operating times. This is possible only if shippers have flexible schedules. In this case, the problem is a *flexible schedule* problem. In this paper we focus on the flexible schedule problem and we will mention that those cases for flexible schedules have the similar results as those for fixed one. Figure 1 provides and illustration of this type of problem.

3.1 Multiple Shipping and Multiple Delivery for Flexible Schedules

We first consider the problem when both transportation schedules (i.e. S_1 and S_2) allow each source to ship cargo several times within its time window (multiple shipping) and each demand point can accept cargo several times within its time window (multiple delivery). Here we take arcs (i, j) between S and T to be the scheduled routes: $(b_i^r, b_i^r + t_{ij})$, $(b_i^r + 1, b_i^r + 1 + t_{ij})$, \dots , $(e_i^r, e_i^r + t_{ij})$, where t_{ij} is the transportation time from source i to TC j and we have discretized time horizon into unit intervals. Arcs between T and D are created similarly.

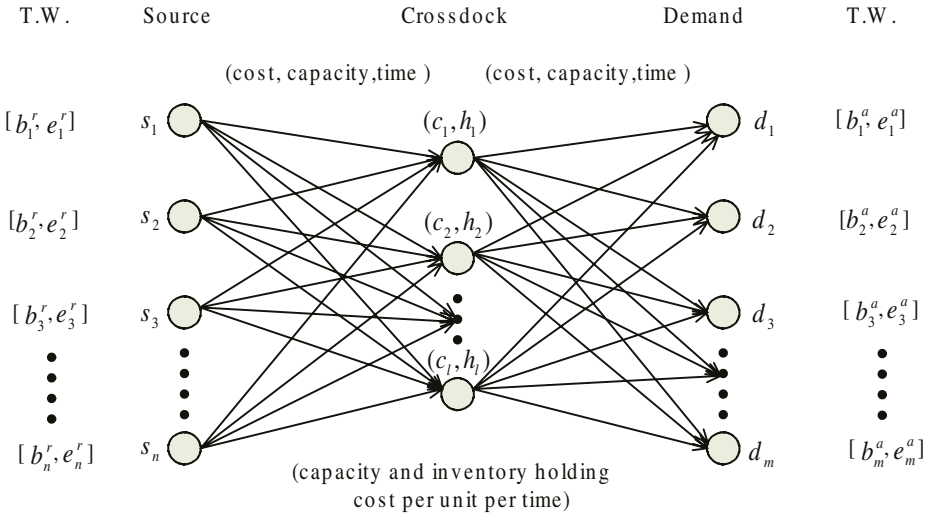


Fig. 1. A flexible schedule problem

When we use Time-expand method to create a duplicate of TC j for each feasible time point (for supply nodes this is a feasible arrival time and for demand nodes this is a feasible deliver time), we should consider transportation time. For example, take two supply nodes s_1 and s_2 , two demand nodes d_1 and d_2 , and a TC node c . Assume s_1 has time window $[7, 14]$ with a transportation time to c equal to 2; for s_2, d_1 and d_2 , take these to be $[9, 12]$ and 1, $[11, 13]$ and 1 and $[10, 12]$ and 2, respectively. Possible arrival time points for s_1 and s_2 are then $\{9, 10, 11, 12, 13, 14, 15, 16\}$ and $\{10, 11, 12, 13\}$, respectively, and possible deliver time points for d_1 and d_2 are $\{10, 11, 12\}$ and $\{8, 9, 10\}$, respectively. Hence the feasible time points for the TC c are $\{9, 10, 11, 12\}$. This fixes the number of duplicates of c to be 4. Note that a feasible time point can be an arrival time, a deliver time or both. We can now give a transformation which can transform this case into a minimum cost flow problem (MCFP) in the following

Transformation A

Step 1: Taking into consideration time windows and transportation times, find all feasible time points for each TC j , and denote N_j be the number of the feasible time points of TC j .

Step 2: Using these feasible time points and original network, construct a new network consisting of five partitions using the following rules:

1. First, generate the third partition. For each TC j , generate N_j duplicates by sorting the time points in an increasing order, connect the duplicates to form a path beginning from the earliest time point to the latest time point one by one with the arcs of cost $\Delta t_i * h_j$, capacity c_j and directed from earlier nodes to later ones, where Δt_i is the length of time interval between the corresponding duplicates.

2. In the second partition, generate a set of feasible supply nodes for each TC j and connect these to j with arcs of zero cost and infinite capacity according the feasibility of the time constraint. Connect those supply nodes which represent the same supply node i ($i = 1, 2, \dots, n$) in the second partition to a new generated node which belongs to the first partition and represents the corresponding supply node i with arcs of cost c_{ij} and capacity k_{ij} .
3. The fourth and last partitions are both made up of demand nodes. And the generation method is similar to the generation of the first and second partitions.
4. Set all the supply nodes i in the first partition and all demand nodes j in the fifth partition to have supply s_i and demand d_j respectively and all other nodes to have zero demand. The direction of each arc between two partitions will be from nodes in the lower-indexed to the higher-indexed partition.

The above method has a pseudo-polynomial time complexity because the duplicate number of each TC depends on the total number of feasible time points. Polynomial time complexity, follows easily from the following Lemmas: Here, inventory refers to cargo which is not shipped out immediately when they reach TCs, i.e., cargo which incurs holding costs at TCs.

Lemma 1 *An optimal solution for the MCFP resulting from Transformation A satisfies the following conditions: (1) Inventory can only have been shipped from source i to TC j at the end time point e_i^r of its time window; (2) The demand point k can only receive inventory at the begin time point b_k^a of its time window, (3) If there are flows from supplier i to demand point k through TC j , then there is only one flow from source i to demand k through one duplicate corresponding to one feasible time point t_{ijk} of TC j , where $t_{ijk} \in [b_{ij}, e_{ij}] \cap [b'_{jk}, e'_{jk}] \neq \phi$ and $[b_{ij}, e_{ij}] = [b_i^r + t_{ij}, e_i^r + t_{ij}]$ and $[b'_{jk}, e'_{jk}] = [b_k^a - t'_{jk}, e_k^a - t'_{jk}]$ and t_{ij} and t'_{jk} are the transportation times from source i to TC j and from TC j to demand k respectively.*

Proof: For (1), assume that in an optimal solution, inventory is shipped from source i to TC j at time points $t_1^r, t_2^r, \dots, t_p^r$, where $t_q^r \in [b_i^r, e_i^r - 1]$, $q = 1, 2, \dots, p$. Then, this inventory cannot be sent out from TC j between $[b_{ij}, e_{ij} - 1]$. Otherwise, we can postpone the shipping time of the inventory to the last time it can be sent out to TCs, where then feasibility still holds, shipping costs remain the same, but inventory cost will be reduced so that the objective value is reduced, which is impossible because of optimality. We find that all this inventory will flow to the last duplicate corresponding to the time point e_i^r . Further, we can also delay shipping this inventory from source i at time e_i^r , for which feasibility holds, shipping costs remain the same but inventory cost will not increase.

For (2), move all inventory received by demand point k during $[b_k^a, e_k^a]$ to the first flow corresponding to time point b_k^a , so that feasibility continues to hold, shipping costs remain the same but inventory cost does not increase.

For (3), if there are several flows from source i to demand k through TC j , then we choose one duplicate corresponding to one feasible time point t_{ijk} of TC j , where $t_{ijk} \in [b_{ij}, e_{ij}] \cap [b'_{jk}, e'_{jk}]$ and merge all the flows into one flow through the duplication t_{ijk} . It is then easy to see that the feasibility holds, inventory

is not affected, and shipping cost remains the same, so that the objective value remains the same. \square

Lemma 2 *The multiple shipping and multiple delivery case can be transformed to a MCFP for which each TC has $m + n$ duplications.*

Proof: By Lemma 1, for each TC j , we only need to consider those duplicates corresponding to time points e_i^r and b_k^a for all $i = 1, 2, \dots, n$; $j = 1, 2, \dots, m$. For $[b_{ij}, e_{ij}] \cap [b'_{jk}, e'_{jk}] \neq \emptyset$, either $e_i^r + t_{ij}$ or $b_k^a - t'_{jk}$ belongs to $[b_{ij}, e_{ij}] \cap [b'_{jk}, e'_{jk}]$. The total duplications are then easily seen to be $m + n$. The method of constructing duplications is different from Transformation A, but the other procedures are in Transformation A. \square

3.2 Single Shipping and Single Delivery for Flexible Schedules

Here we consider the case when there can be only a single shipment out of each supply point within its time window, and, similarly, where each customer can accept cargo only once within its time window. In such case, there can be several arcs for each supply point, of which only one can be used. Similarly, there can be several paths in S_2 leading to a demand point, but where only one will be used. A single shipping strategy can be preferred by a shipper as it requires less setup costs and fixed costs related to shipping each consignment through one or more shipping agents. The following theorems show that this problem is **NP**-complete in the strong sense, even if time windows and capacities are relaxed. These hardness results imply no polynomial algorithm exists for the single shipping problem.

THEOREM 1: *The flexible schedule single shipping and single delivery problem is **NP**-complete in the strong sense, even if supply node time windows and transportation capacities are relaxed. The same hardness is valid for the problem of finding a feasible solution*

Proof: We give a reduction from the **3-PARTITION** problem. Here, our problem is: Does there exist a feasible schedule that achieves the objective value no greater than K ? From an arbitrary instance of **3-PARTITION**, consider the following polynomial reduction to an instance of our problem. We have $3w$ suppliers of S , and w customers $D = \{1, 2, \dots, w\}$. For each $i \in S$, let $s(i)$ be its provision, while for each $j \in D$, let the demand be B with time window $[j, j]$. Exactly one TC c with capacity B and unit holding cost for per unit cargo per unit time exists to link suppliers and customers, and both time delay and shipping cost are zero for all routes. Let K be zero. We now show that a feasible schedule of objective value equal to zero exists if and only if the **3-PARTITION** has a feasible solution. On one hand, if **3-PARTITION** has a feasible solution of S_1, \dots, S_w , then we ship all cargo provided by suppliers in S_i to c at time j , which satisfies the demands B for each j within its time window $[j, j]$, where $j = 1, 2, \dots, w$. It is easy to verify that such a schedule is feasible and its objective value is zero. On the other hand, if a feasible schedule exists whose corresponding objective value is zero, then we can construct a partition by setting S_j to

be the subset of $i \in S$ with positive flow to c at time j , for $1 \leq j \leq w$. Because the objective value is zero and the demands are B at time j , we can assume that there is no inventory before time j . We then have $\sum_{i \in S_j} s(i) \geq B$, where if $\sum_{i \in S_j} s(i) > B$, then $\sum_{i \in S_j} s(i) - B$ units of inventory will remain in the TC incurring (non-zero) cost. This contradicts the zero objective value. Since $B/4 < s(i) < B/2$ for $i \in S$, we have $|S_j| = 3$ and because of the single shipping constraint, we know that S_1, \dots, S_w is a feasible partition for the instance of **3-PARTITION** and this completes the proof. \square

3.3 The Remaining Cases for Flexible Schedules

The cases, for single shipping multiple delivery and multiple shipping single delivery are both **NP**-complete in the strong sense, even if supply point time windows and transportation capacities are relaxed. The same hardness is valid for the problems for finding a feasible solution. As the proof is similar to those given in Section 3.2, we omit these here. Another important result is that for the case multiple shipping and multiple delivery without inventory capacity in TCs, we can enhance our model to deal with continuous time, and similar transformation method can be used to solve it in polynomial time complexity.

3.4 Cases for Fixed Schedules and Mixed Schedules

When the schedules are fixed, we still have the same result for each case as that for flexible schedules. We also study the mixed schedules that are a combination of a fixed schedule and a flexible one. These conditions can occur when, say, a fixed schedule is the only one available to shippers or the schedule preferred by shipper. By working on such kind of problems, we also find that each case for mixed schedules still has the same result as that for flexible one. Because the limitation of the paper length and the ideas and methods for fixed schedules and mixed one are similar to the flexible one, we omit the details.

4 Other Applications

The transformation techniques used for transshipment problems developed here can be used for other problems. Here we only describe how to apply them to the extended problem with penalties on suppliers. It is common in practice to penalize late shipments which can incur significant costs to the entire supply chain. In the following, we show the techniques developed are easily adapted to penalties imposed on shipments made outside of supplier time windows. To do this, we add penalties directly to the shipping costs in the S_1 which we represent as piecewise functions where shipping costs $c_{ij,t_b t_e}$ for $i \in S$ and $j \in T$ can be defined as follows:

$$c_{ij,t_b t_e} = \begin{cases} w_{ij,t_b t_e} + \alpha_{ij,t_b t_e} & \text{if } t_b < b_i^r \\ w_{ij,t_b t_e} & \text{if } t_b \in [b_i^r, e_i^r] \\ w_{ij,t_b t_e} + \beta_{ij,t_b t_e} & \text{if } t_b > e_i^r \end{cases}$$

where $w_{ij,t_b t_e}$ are original unit shipping costs, and $\alpha_{ij,t_b t_e}$ and $\beta_{ij,t_b t_e}$ are positive and represent unit earliness and tardiness penalties respectively. Under these assumptions, we can apply the techniques already developed to the following cases.

4.1 Flexible Schedule Problems with Penalties

Here we only discuss the cases for flexible schedule, and those cases for fixed schedules and mixed one have the same results by using the similar ideas and methods. In the multiple shipping and multiple delivery case, each path from supply to demand through a TC is now feasible. Here, paths can be of three types: early, on-time and tardy, where early paths include paths which can only begin earlier than the time window, on-time paths are those which begin within the time window, and tardy paths are those paths which begin later than the time window. As mentioned before, applying Lemma 2, we have improved the Transformation A to give a polynomial solution. Now we modify this improved Transformation A to solve the penalty case. In this case, if $b_{ij} > e'_{jk}$ (b_{ij} and e'_{jk} are defined as in Section 3.1), we can choose e'_{jk} as a extra feasible time point belonging to a early path, and the total number of such early feasible time points is no more than m . We then find that, similar to the previous analysis, the number of duplicates of each TC is no more than $2m+n$, where the additional m duplicates arise from early feasible time points. We connect supply nodes in the second partition to the duplicates of each TC in the third partition according to the following rules: for the on-time path, i.e. $[b_{ij}, e_{ij}] \cap [b'_{jk}, e'_{jk}] \neq \emptyset$, we connect supply i to the duplicate representing time point e_{ij} or b'_{jk} ; for a tardy path, i.e. $e_{ij} < b'_{jk}$, connect supply i to the duplicates representing time point e_{ij} and b'_{jk} respectively; and for early path, i.e. $b_{ij} > e'_{jk}$, connect supply i to the duplicate representing time point e'_{jk} . And those added arcs have infinite capacity and costs corresponding to their path types. After above modification, we can use the modified transformation method to solve this case.

4.2 The Remaining Cases for All Problems

The remaining cases for all problems in section 3 are strongly **NP**-complete problems and the results of their corresponding supplier penalties problems continue to hold with similar proofs.

5 Summary

We have studied transshipment through crossdocks which includes inventory and time window considerations. The objectives for this study has been twofold: (1) to find cost optimal schedules in transshipment networks with time-dependent supply and demand which satisfy transportation schedule constraints, and (2) to meet the primary objective of minimizing inventory at crossdocks within the

context of (1). The techniques used are easily implemented, where for polynomially solvable cases, we provided transformation techniques which reduce the problem to a flow problem. Other cases were shown to be **NP**-complete in strong sense.

Acknowledgement

The third-named author acknowledges funding support from the Wharton-SMU Research Center, Singapore Management University.

References

1. Ahuja, R.K., Magnanti, T.L., and Orlin, J.B., 1993, Network Flows, Prentice-Hall, Upper Saddle River, New Jersey
2. Aronson, J.E., 1989, A survey on dynamic network flows, *Annals of Operations Research*, 20, pp. 1-66
3. Bartholdi, John J. III and Gue, Kevin R, 2002, Reducing labor costs in an LTL crossdocking terminal, *Operations Research*, Vol. 48, No. 6, pp. 823-832
4. Fisher, M.L., Hammond, J.H., Obermeyer, W.R., and Raman, A., 1994, Making supply meet demand in an uncertain world, *Harvard Business Review*, May-June, pp. 83-93
5. Herer, Y.T. and M. Tzur, 2001, The Dynamic Transshipment Problem, *Naval Research Logistics*, 48, pp. 386-408
6. Hoppe, B. and Tardos, E., 1995, The quickest transshipment problem, *SODA: ACM-SIAM Symposium on Discrete Algorithms*
7. Kamarkar, U.S., 1987, The Multilocation Multiperiod Inventory Problem: Bounds and Approximations. *Management Science*, 33(1), pp. 86-94.
8. Karmarkar, U.S. and Patel, N.R., 1977, The One-Period N-Location Distribution Problem, *Naval Research Logistics*, 24, pp. 559-575
9. Napolitano, M., 2000, Making the move to crossdocking, *Warehousing Education and Research Council*
10. Robinson, L.W., 1990, Optimal and approximate policies in multiperiod, multilocation inventory models with transshipments, *Operations Research*, 38, pp. 278-295.
11. Rudi, N., Kapur, S., Pyke, D. F., 2001, A two-location inventory model with transshipment and local decision making, *Management Science*, 47 pp. 1668-1680.
12. Simchi-Levi, D., Kaminsky, P., Simchi-Levi, E., 2003, *Designing and Managing the Supply Chain*, 2ed. McGraw-Hill - Irwin Publishers

Approximated Vertex Cover for Graphs with Perfect Matchings

Tomokazu Imamura¹, Kazuo Iwama¹, and Tatsuie Tsukiji²

¹ School of Informatics, Kyoto University, Kyoto 606-8501, Japan
{imamura,iwama}@kuis.kyoto-u.ac.jp

² Department of Information Science, Tokyo Denki University,
Hatoyama, Saitama 350-0394, Japan
tsukiji@j.dendai.ac.jp

Abstract. Chen and Kanj considered the VERTEX COVER problem for graphs with perfect matchings (VC-PM). They showed that: (i) There is a reduction from general VERTEX COVER to VC-PM, which guarantees that if one can achieve an approximation factor of less than two for VC-PM, then one can do so for general VERTEX COVER as well. (ii) There is an algorithm for VC-PM whose approximation factor is given as $1.069 + 0.069\bar{d}$ where \bar{d} is the average degree of the given graph. In this paper we improve (ii). Namely we give a new VC-PM algorithm which greatly outperforms the above one and its approximation factor is roughly $2 - \frac{6.74}{\bar{d}+6.28}$. Our algorithm also works for graphs with “large” matchings although its approximation factor is degenerated.

1 Introduction

Everybody agrees that VERTEX COVER is one of the most important combinatorial-optimization problems, which has a variety of both practical and theoretical applications. Another important aspect of this problem is its approximability: There has been a well-known, straightforward algorithm which achieves an approximation factor of two, but in spite of much effort, no one has succeeded in providing an upperbound of $2 - \varepsilon$ or a lowerbound of two. (The current best lowerbound is 1.3606 [6]). Under the assumption that achieving a general upperbound of $2 - \varepsilon$ is quite hard, our common approach is to investigate several restricted cases which give us some insight on the general goal and/or which are important in practice. In fact there is a large literature along this line (e.g., [11], [3], [10], [1], [12], [4]).

In [4], Chen and Kanj considered this problem for graphs with perfect matchings, which they called VC-PM. They gave an interesting reduction from the general VERTEX COVER to VC-PM which somewhat preserves the approximation factor. Using this reduction, they proved that if one can achieve an approximation factor of less than two for VC-PM, then one can do so for the general problem as well. Thus, for the purpose of breaking the barrier of two, it is enough to attack VC-PM. Unfortunately, however, the specific VC-PM algorithm they

presented in [4] does not seem to have much gain in this sense. Their approximation factor, given as $1.069 + 0.069\bar{d}$, becomes two when $\bar{d} = 13.5$, where \bar{d} is the average degree of the given graph.

Our contribution. We give a new algorithm for VC-PM, whose approximation factor is illustrated in Fig.1. One can see that it is much better than that of Chen and Kanj's and also outperforms the well-known algorithm by Halldórsson and Radhakrishnan when it is run for graphs with perfect matchings (the latter algorithm is actually for obtaining an independent set, but can also be used for the current problem). Our algorithm also works for graphs with large matchings (whose size is smaller than $n/2$) although the approximation factor is degenerated.

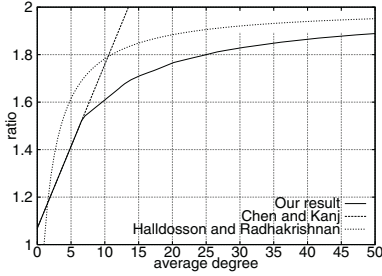


Fig. 1. Comparison to the previous results

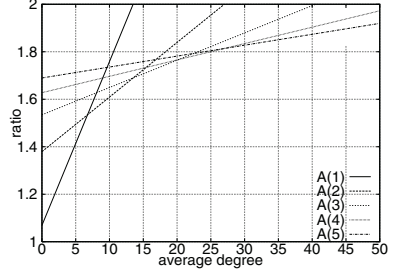


Fig. 2. Approximation factor of VCPM-Apx(w) for each w

The basic idea of the new algorithm is to use the WEIGHTED MAX-2SAT and adjust the weighting function according to the average degree of the instance graph. Thus our algorithm A is actually a sequence of algorithms $A(1), A(2), \dots, A(w), \dots$, parameterized by w (the weight for matching edges). For each w , $A(w)$ is better than any other $A(w')$ in some range of \bar{d} as illustrated in Fig.2.

Previous work. For general VERTEX COVER, finding any maximal matching in the given graph and choosing all vertices in the matching yields a factor-2 vertex cover. However, no approximation algorithm of factor $2 - \varepsilon$ is known for any constant $\varepsilon > 0$. The current best approximation algorithms are those found by Bar-Yehuda and Even [2] and Monien and Speckenmeyer [14], whose approximation factor is $2 - \frac{\log \log n}{2 \log n}$, and one found by Halperin [10] which achieves a factor of $2 - (1 - o(1)) \frac{2 \ln \ln n}{\ln n}$. There has been no further progress on this general bound for about two decades.

However, for the restricted version of the problems, many good algorithms are invented. For graphs whose maximum degree is bounded by Δ , Hochbaum [11] presented $2 - \frac{2}{\Delta}$ factor using a coloring method. Exploiting Halldórsson and Radhakrishnan's [9] approximation algorithm for the Independent Set problem, one can obtain factor of $2 - \frac{\log \Delta + O(1)}{\Delta}$. Using semidefinite programming, Halperin [10] asymptotically improved these results by presenting an approximation algo-

rithm of factor $2 - (1 - o(1)) \frac{2 \ln \ln \Delta}{\ln \Delta}$. For small Δ , Berman and Fujito [3] achieved a factor of $2 - \frac{5}{\Delta+3}$. For planar graphs this problem admits PTAS [1]. For ε -dense graphs, that is, for those graphs where each vertex has at least $\varepsilon|V|$ neighbors, Karpinski and Zelikovsky [12] showed that there is an approximation algorithm of factor $2/(1 + \varepsilon)$. For graphs whose average degree can be represented as $\varepsilon|V|$, Karpinski and Zelikovsky [12] also showed that it is approximable within a factor of $2/(2 - \sqrt{1 - \varepsilon})$.

For VC-PM, other than the algorithm in [4] whose approximation factor is $1.069 + 0.069\bar{d}$, only the algorithm by Halldórsson and Radhakrishnan [9] seems competitive in terms of the approximation factor parameterized by the average degree. Its approximation factor is given as $2 - 5/(2\bar{d} + 3)$.

For the negative direction, Håstad [10] showed that it is NP-hard to approximate VERTEX COVER within a factor of less than $7/6 = 1.1666$. Dinur and Safra [6] improved this lowerbound to $10\sqrt{5} - 21 = 1.3606$. Even for graphs whose maximum degree is bounded by $\Delta \geq 3$, this problem is APX-complete and for large Δ it is NP-hard to approximate within 1.3606 [5]. As a most recent result, Khot and Regev [13] showed that, if a certain conjecture about the PCP theory is true, we can prove the nonexistence of $2 - \varepsilon$ factor approximation algorithm for Vertex Cover. For VC-PM, using the reduction introduced by Chen and Kanj [4], we can easily obtain a lowerbound of $5\sqrt{5} - 10 = 1.180$.

2 Notations and Definitions

We employ standard terminologies in graph theory. Let $G = (V, E)$ be a graph. For a subset $W \subseteq V$, the *induced graph* $G(W) = (W, E(W))$ is the subgraph of G induced by W , i.e. $E(W) = \{(u, v) : u, v \in W \text{ and } (u, v) \in E(G)\}$. We denote by \bar{W} the complement of $W \subseteq V$, that is $V - W$. For every vertex $v \in V$, $N_G(v)$ is the set of vertices adjacent to v . The *degree* $d_G(v)$ of v in G is the size of $N_G(v)$, namely $d_G(v) = |N_G(v)|$. The *average degree* \bar{d} of a graph G is defined as $\bar{d}(G) = 2|E|/|V|$.

VERTEX COVER and INDEPENDENT SET are fundamental NP-complete problems [8]. A set $C \subseteq V$ of vertices is a (*vertex*) *cover* of a graph G if every edge of G has at least one endpoint in C . Let $Opt(G)$ be the size of a smallest vertex cover of G . VERTEX COVER is a computational problem that asks for an optimal-size vertex cover of a given graph. Although VERTEX COVER is NP-hard, polynomial-time algorithms are known that output a vertex cover C of any given graph G such that $|C|/|Opt(G)| \leq r$, where $r \geq 1$ is a real number called an *approximation factor* of the algorithm. A set I of vertices of a graph G is an *independent set* of G if every edge of G has at most one endpoint in I . An algorithm for INDEPENDENT SET is said to achieve a factor r if it outputs an independent set I of a given graph G such that $Opt_{IS}(G)/|I| \leq r$, where $Opt_{IS}(G)$ is the size of a largest independent set of G . If a given graph G satisfies $Opt(G) \geq \frac{|V(G)|}{2}$, then one can use an r -approximation algorithm for INDEPENDENT SET as a $(2 - 1/r)$ -approximation algorithm to obtain a vertex cover of G , by letting the algorithm find an independent set of G and output its complement as a vertex cover of G .

This paper studies VERTEX COVER WITH PERFECT MATCHING (VC-PM), which is a version of VERTEX COVER where the instance graph is promised to have a perfect matching [4].

We use an approximation algorithm for WEIGHTED MAX-2SAT. Fix a set X of Boolean variables. An *assignment* σ to X is a mapping from X to $\{\text{TRUE}, \text{FALSE}\}$. A *CNF-formula* F is a Boolean formula consisting of a conjunction of disjunctions of literals in $\{x, \bar{x} : x \in X\}$, where each disjunction is called a *clause* of F . A CNF-formula is *weighted* if each clause is assigned a real number, called the *weight* of the clause. The weight $|\sigma|_F$ of an assignment σ to X with respect to F is the sum of weights of the clauses satisfied by σ . Let $\text{Opt}(F)$ be the weight of an heaviest assignment with respect to F . An r -approximation algorithm of WEIGHTED MAX-2SAT computes an assignment σ of a given formula F such that $\text{Opt}(F)/|\sigma|_F \leq r$. An assignment σ is said *maximal* if for each variable x such that $\sigma(x) = \text{FALSE}$, flipping $\sigma(x)$ from FALSE to TRUE decreases $|\sigma|_F$. We abbreviate $|\sigma|_F$ as $|\sigma|$, if there is no confusion.

3 Our Algorithm

Let $G = (V, E)$ be an instance graph of VC-PM, so G is assumed to have a perfect matching. The goal of VC-PM is to obtain a vertex cover C of G such that $|C| = \text{Opt}(G)$. We denote $n = |V|$ and $m = |E|$.

3.1 Basic Ideas

For finding an approximate feasible solution C of G , Chen and Kanj has taken the following strategy (see Lemma 4.1 of [4] for details): Let

$$F_G = \bigwedge_{(u,v) \in E} (x_u \vee x_v) \wedge \bigwedge_{(u,v) \in M} (\bar{x}_u \vee \bar{x}_v).$$

Thus, F_G is an unweighted CNF-formula having variables in a set $X_G = \{x_u : u \in V\}$, and having $m + n/2$ clauses. Regarding this formula as a MAX-2SAT instance, they apply a MAX-2SAT approximation algorithm to obtain some assignment σ to X_G . Without loss of generality this assignment is assumed to be maximal. Then, they give the following vertex set of G : $\mathcal{T}_\sigma = \{v \in V | \sigma(x_v) = \text{TRUE}\}$. Let us call the m clauses $(x_u \vee x_v)$ such that $(u, v) \in E$ the *positive clauses* of F_G and the other $n/2$ clauses, $(\bar{x}_u \vee \bar{x}_v)$ such that $(u, v) \in M$, the *negative clauses*. Obviously \mathcal{T}_σ covers all edges in the graph G if and only if the underlying assignment σ to X_G satisfies all positive clauses of the formula F_G . In addition, every maximal assignment satisfies all positive clauses. Thus, \mathcal{T}_σ is a vertex cover of G and their algorithm outputs it. On the other hand, the negative clauses of F_G play the role of bounding the covering size $|\mathcal{T}_\sigma|$. As a whole, if we find an assignment which satisfies as many clauses as possible, then it turns out to be a “good” assignment for our objective.

Here comes the major difference between our new algorithm and the Chen and Kanj’s: We also use the same CNF-formula F_G , but as an *weighted* MAX-2SAT instance. Each positive clause has weight of one as before but each negative

clause has weight of w which may be greater than one. Intuitively adopting a larger w may derive a smaller $|\mathcal{T}_\sigma|$, namely, we can make the role of negative clauses more powerful. But, unfortunately, this setting may cause a drawback that a maximal assignment σ may not satisfy some positive clauses of F_G . In order to remedy this drawback our algorithm finds another vertex cover $C_{\mathcal{F}}$ of the subgraph induced by the vertices in $\mathcal{F}_\sigma = \{v \in V | \sigma(x_v) = \text{FALSE}\}$. Obviously the union $\mathcal{T}_\sigma \cup C_{\mathcal{F}}$ covers the original graph G and thus the algorithm outputs it as a vertex cover of G .

3.2 New Algorithm

Figure 3 presents a formal description of our algorithm, which we call as VCPM-Apx(w), VCPM-Apx(1) coincides the algorithm by Chen and Kanj, because $G(\mathcal{F}_\sigma)$ contains no edge (by Lemma 1) and so the empty set can be given as a covering $C_{\mathcal{F}_\sigma}$ of $G(\mathcal{F}_\sigma)$ in step 6.

1. Input a graph $G = (V, E)$ with a perfect matching M .
2. Construct a formula F_G .
3. Solve F_G and let σ be the resulting approximate solution.
4. Refine σ to meet maximality.
5. Set $\mathcal{T}_\sigma = \{v \in V | \sigma(x_v) = \text{TRUE}\}$ and $\mathcal{F}_\sigma = \{v \in V | \sigma(x_v) = \text{FALSE}\}$
6. Find a vertex cover $C_{\mathcal{F}}$ for $G(\mathcal{F}_\sigma)$.
7. Output $\mathcal{T}_\sigma \cup C_{\mathcal{F}}$.

Fig. 3. VCPM-Apx(w)

In order to complete this algorithm we need to specify a weighted MAX-2SAT algorithm for step 2 and a VERTEX COVER algorithm for step 5. For the former one we adopt one of Feige and Goemans [7], celebrated to achieve the best known approximation factor $r = 1.0741 \dots$. In the next section we show that, if an assignment σ is maximal then the maximum degree of $G(\mathcal{F}_\sigma)$ is bounded by $w - 1$. For the latter one, we therefore adopt one of approximation algorithms that perform better for bounded degree graphs. In this paper we adopt the greedy INDEPENDENT SET algorithm. Beginning from $G(\mathcal{F}_\sigma)$, the greedy algorithm picks a minimum-degree vertex and removes the adjacent vertices from the current graph. The greedy algorithm finally reaches to an independent set I and outputs its complement $\mathcal{F}_\sigma - I$ as a covering $C_{\mathcal{F}_\sigma}$ of $G(\mathcal{F}_\sigma)$. If the maximum degree of $G(\mathcal{F}_\sigma)$ is bounded by $w - 1$, then we have $|C_{\mathcal{F}_\sigma}| \leq (1 - 1/w)|\mathcal{F}_\sigma|$. We use this upperbound to derive an upper bound of the approximation factor of VCPM-Apx(w). See Lemmas 5 and 6 for details.

Remark. In Section 1 we have introduced much better approximation factors of VERTEX COVER. These algorithms can improve the approximation factor $|C_{\mathcal{F}_\sigma}|/\text{Opt}(G(\mathcal{F}_\sigma))$ if they are applied to $G(\mathcal{F}_\sigma)$, but we actually need better upper bounds of both $|C_{\mathcal{F}_\sigma}|$ and $|C_{\mathcal{F}_\sigma}|/\text{Opt}(G(\mathcal{F}_\sigma))$ to get a better upper bound

of the objective factor $|\mathcal{T}_\sigma \cup C_{\mathcal{F}_\sigma}|/Opt(G)$. In addition, the upperbound $|C_{\mathcal{F}_\sigma}| \leq (1 - 1/w)|\mathcal{F}_\sigma|$ is optimal when the graph $G(\mathcal{F}_\sigma)$ consists of a disjoint union of complete graphs of size $w - 1$.

4 Approximation Factors of Our Algorithms

In this section we give approximation factors of our algorithm VCPM-Apx(w). Let $G = (V, E)$ be an instance graph and F_G be the associated formula. Let σ be an assignment to $\{x_v : v \in V\}$ obtained by an r -approximation algorithm applied to F_G and refined to meet maximality. Let $\mathcal{T}_\sigma = \{v \in V | \sigma(x_v) = \text{TRUE}\}$ and $\mathcal{F}_\sigma = \{v \in V | \sigma(x_v) = \text{FALSE}\}$. Let $C_{\mathcal{F}}$ be a cover of $G(\mathcal{F})$ obtained by applying the greedy algorithm to $G(\mathcal{F})$. Let $C = \mathcal{T}_\sigma \cup C_{\mathcal{F}}$. Let σ_C be the associated assignment, i.e. $\sigma_C(x_v) = \text{TRUE} \Leftrightarrow v \in C$. Let

$$\delta = (|\sigma| - |\sigma_C|)/n.$$

Lemma 1. *The maximum degree of $G(\mathcal{F}_\sigma)$ is bounded by $w - 1$.*

Proof. Assume that there is a vertex v in \mathcal{F}_σ whose degree in $G(\mathcal{F}_\sigma)$ is greater than or equal to w . Since all positive clauses $(x_u \vee x_v)$ such that $(u, v) \in E(G(\mathcal{F}_\sigma))$ and $u \in \mathcal{F}_\sigma$ are unsatisfied by σ , if one switches $\sigma(x_v)$ from FALSE to TRUE then at least w positive clauses are newly satisfied. On the other hand this change of σ may turn the truth value of the negative clause $(\bar{x}_u \vee \bar{x}_v)$ with $(u, v) \in M$ to FALSE hence we may lose weight by w . As a whole $|\sigma|$ doesn't decrease by that change. This contradicts the maximality of σ .

Lemma 2. *Let G be a graph, C be any vertex cover of G and σ_C be the associated assignment, then*

$$|C| = n + \frac{1}{w}(m - |\sigma_C|).$$

Proof. Since C is a cover of G , the assignment σ_C satisfies all m positive clauses of F_G , so $|\sigma_C| - m$ is equal to the sum of weights of the negative clauses satisfied by σ_C . On the other hand, either $u \in C$ or $v \in C$ is met because $(u, v) \in M$ and C is a vertex cover. In addition, $u \notin C$ or $v \notin C$, since $(\bar{x}_u \vee \bar{x}_v)$ is satisfied by σ_C . Thus, it follows that the number of negative clauses of F_G satisfied by σ_C is equal to the number of vertices uncovered by C , that is $n - |C|$. Each of these negative clauses has weight w , so we obtain $|\sigma_C| - m = w(n - |C|)$. The lemma follows by expelling $|C|$ from it.

Lemma 3.

$$Opt(G) \geq n + \frac{1}{w}(m - Opt(F_G)).$$

Proof. Let C be a minimum vertex cover of the instance graph G and let σ_C be the associated assignment. By lemma 2, it follows that $Opt(G) = |C| = n + (m - |\sigma_C|)/w$. Since $Opt(F_G)$ is the largest weight with respect to the formula F_G , we obtain $Opt(F_G) \geq |\sigma_C|$. Therefore $n + (m - |\sigma_C|)/w \geq n + \frac{1}{w}(m - Opt(F_G))$. The lemma follows.

Lemma 4.

$$\frac{|C|}{\text{Opt}(G)} \leq 1 + \frac{2}{w}\delta + \frac{r-1}{wr}(\bar{d}(G) + w).$$

Proof. The approximation factor of C is bounded from above as follows:

$$\begin{aligned} \frac{|C|}{\text{Opt}(G)} &= \frac{n + (m - |\sigma_C|)/w}{\text{Opt}(G)} \\ &= \frac{n + (m + \delta n - |\sigma|)/w}{\text{Opt}(G)} \\ &\leq \frac{n + (m + \delta n - \text{Opt}(F_G)/r)/w}{\text{Opt}(G)} \\ &= \frac{n + \frac{1}{w}(m - \text{Opt}(F_G))}{\text{Opt}(G)} + \frac{\frac{1}{w}(\delta n + \frac{r-1}{r}\text{Opt}(F_G))}{\text{Opt}(G)} \\ &\leq 1 + \frac{\frac{1}{w}(\delta n + \frac{r-1}{r}(m + wn/2))}{n/2} \\ &= 1 + \frac{2\delta}{w} + \frac{r-1}{wr}(\bar{d} + w). \end{aligned}$$

The first equality follows from lemma 2, the second equality follows from the definition of δ , the third inequality follows from a factor r of approximation to obtain σ , the fifth inequality follows from lemma 3, the fact that $\text{Opt}(F_G) \leq m + wn/2$, and the assumption that G has a perfect matching and so $\text{Opt}(G) \geq n/2$.

Now we have obtained an approximation factor of C . But, in the right hand side of this inequality, we have δ whose value is unknown. So this estimation cannot be used to obtain an approximation factor of our algorithm, because when δ is large this upper bound exceeds 2 which is the trivial upper bound for general VERTEX COVER. Instead, in the following we show another estimation of C that can be used when δ is large.

Lemma 5.

$$\frac{|\mathcal{F}_\sigma| - |C_\mathcal{F}|}{|\sigma| - |\sigma_C|} \geq \frac{2}{w(w-1)}.$$

Proof. We apply the greedy algorithm to $G(\mathcal{F}_\sigma)$. Let $f = |\mathcal{F}_\sigma - C_\mathcal{F}|$. Let v_1, \dots, v_f be the independent vertices picked by the greedy algorithm in this order. Let $C_0 = \emptyset$, and $C_i = C_{i-1} \cup (N_G(v_i) \cap \mathcal{F}_\sigma)$, for each $1 \leq i \leq f$. Thus $C_f = C_\mathcal{F}$. Let $E_0 = E(G(\mathcal{F}_\sigma))$, and E_i be the set of edges in $G(\mathcal{F}_\sigma)$ that are adjacent to none of the vertices in C_i . Let $H_0 = G(\mathcal{F}_\sigma)$ and $H_i = (\mathcal{F}_\sigma, E_i)$, for each $1 \leq i \leq f$. Let σ_i be the assignment to $\{x_v : v \in V(G)\}$ such that $\sigma_i(x_v) = \text{TRUE} \Leftrightarrow v \in \mathcal{T}_\sigma \cup C_i$. Thus $\sigma_f = \sigma_C$.

Then, $C_i = C_{i-1} \cup N_{H_{i-1}}(v_i)$ for each $1 \leq i \leq f$. Particularly, $\{v_1, \dots, v_f\} \cap C_\mathcal{F} = \emptyset$, hence $|\mathcal{F}_\sigma| - |C_\mathcal{F}| \geq f$.

Let $\Delta = w-1$. Lemma 1 shows that $d_{G(\mathcal{F}_\sigma)} \leq \Delta$. We claim that $|\sigma_{i-1}| - |\sigma_i| \leq \Delta(\Delta + 1)/2$ for each $1 \leq i \leq f$.

Case 1: $H_{i-1}(N_{H_{i-1}}(v_i) \cup \{v_i\})$ is the Δ -complete graph. The assignment σ_i is obtained from σ_{i-1} by flipping the truthvalue $\sigma_{i-1}(x_v)$ for each $v \in N_{H_{i-1}}(v_i)$ from FALSE to TRUE. Thus, for each edge (u, v) of $H_{i-1}(N_{H_{i-1}}(v_i) \cup \{v_i\})$, the positive clause $(x_u \vee x_v)$ is unsatisfied by σ_{i-1} but satisfied by σ_i . Since $H_{i-1}(N_{H_{i-1}}(v_i) \cup \{v_i\})$ is the Δ -complete graph, it contains $\Delta(\Delta+1)/2$ edges. So, the contribution from the positive clauses of F_G to $|\sigma_{i-1}| - |\sigma_i|$ is at most $-\Delta(\Delta+1)/2$. On the other hand, there are at most $|N_{H_{i-1}}(v_i)| = \Delta$ edges $(u, v) \in M$ such that $\{u, v\} \cap N_{H_{i-1}}(v_i) \neq \emptyset$, and the corresponding negative clauses $(\bar{x}_u \vee \bar{x}_v)$ may be satisfied by σ_{i-1} but unsatisfied by σ_i . So, the contribution from the negative clauses of F_G to $|\sigma_{i-1}| - |\sigma_i|$ is at most $\Delta w = \Delta(\Delta+1)$. As a whole, we have $|\sigma_{i-1}| - |\sigma_i| \leq \Delta(\Delta+1) - \Delta(\Delta+1)/2 = \Delta(\Delta+1)/2$.

Case 2: $H_{i-1}(N_{H_{i-1}}(v_i) \cup \{v_i\})$ is not the Δ -complete graph. Let $\ell = d_{H_{i-1}}(v_i)$. Since v_i is picked by the greedy algorithm from $V(H_{i-1})$, $d_{H_{i-1}}(u) \geq \ell$ for every vertex $u \in N_{H_{i-1}}(v_i)$. In addition, $H_{i-1}(N_{H_{i-1}}(v_i) \cup \{v_i\})$ is not the Δ -complete graph, so $\ell < \Delta$. There are at least $\ell(\ell+1)/2$ edges of H_{i-1} that are adjacent to some vertex in $N_{H_{i-1}}(v_i)$, whose associated positive clauses of F_G are unsatisfied by σ_{i-1} and satisfied by σ_i . So, the contribution from the positive clauses of F_G to $|\sigma_{i-1}| - |\sigma_i|$ is at most $-\ell(\ell+1)/2$. On the other hand, as in Case 1, there are at most $\ell = |N_{H_{i-1}}(v_i)|$ matching edges whose associated negative clauses may be satisfied by σ_{i-1} and unsatisfied by σ_i . So, the contribution from the negative clauses of F_G to $|\sigma_{i-1}| - |\sigma_i|$ is at most $\ell w = \ell(\Delta+1)$. As a whole, we have $|\sigma_{i-1}| - |\sigma_i| \leq \ell(\Delta+1) - \ell(\ell+1)/2 = \ell(2\Delta - \ell + 1)/2$, which is smaller than $\Delta(\Delta+1)/2$.

Now, we have $|\sigma_{i-1}| - |\sigma_i| \leq \Delta(\Delta+1)/2$ for every $1 \leq i \leq f$, so $|\sigma| - |\sigma_C| = \sum_{i=1}^f |\sigma_{i-1}| - |\sigma_i| \leq f\Delta(\Delta+1)/2$. On the other hand, $|\mathcal{F}_\sigma| - |C_\mathcal{F}| \geq f$, thus

$$\frac{|\mathcal{F}_\sigma| - |C_\mathcal{F}|}{|\sigma| - |\sigma_C|} \geq \frac{2}{\Delta(\Delta+1)} = \frac{2}{w(w-1)}.$$

Lemma 6.

$$\frac{|C|}{\text{Opt}(G)} \leq 2 - \frac{4\delta}{w(w-1)}.$$

Proof. Since $C = \mathcal{T}_\sigma \cup C_\mathcal{F}$,

$$|C| \leq n - (|\mathcal{F}_\sigma| - |C_\mathcal{F}|) = n - \frac{2\delta n}{w(w-1)}$$

where the last equality follows from Lemma 5. Since the instance graph G has a perfect matching, $\text{Opt}(G) \geq n/2$, so $\frac{|C|}{\text{Opt}(G)} \leq \frac{n - \frac{2\delta n}{w(w-1)}}{n/2} = 2 - \frac{4\delta}{w(w-1)}$.

The two bounds in lemmas 4 and 6 are oppositely related as functions of δ . Their combination thus provides the following approximation factor of VCPM-Apx(w).

Theorem 1. *VCPM-Apx(w) achieves an approximation factor*

$$2 - 2 \frac{(1-r)\bar{d} + w}{rw(w+1)}$$

for graphs of average degree \bar{d} .

Proof. We choose the better one of the two approximation factors given by lemmas 4 and 6, respectively. That is, we set

$$\alpha = \frac{w(w-1)}{2(w+1)} \left(1 - \frac{r-1}{rw} (\bar{d} + w) \right)$$

and use lemma 4 if $\delta \leq \alpha$, else we use Lemma 6 and obtain

$$\frac{|C|}{\text{Opt}(G)} \leq 2 - 2 \frac{(1-r)\bar{d} + w}{rw(w+1)}.$$

This theorem gives a series of approximation factors of VCPM-Apx(w) for $w = 2, 3, 4, \dots$ (see Figure 2), so we take their lower envelope. Particularly, since $r > 1$, choosing appropriate w according to \bar{d} gives us a factor of less than two for all \bar{d} . In more precise, for \bar{d} such that $(w-1)/2(r-1) \leq \bar{d} < w/2(r-1)$, VCPM-Apx(w) works better than VCPM-Apx(w') for any $w' \neq w$. So we combine VCPM-Apx(w) accordingly, adopting VCPM-Apx(w) if G meets $(w-1)/2(r-1) \leq \bar{d}(G) < w/2(r-1)$. It achieves an approximation factor

$$2 - \frac{2}{r} \left(2(r-1)\bar{d} + 1.0 - \sqrt{(2(r-1)\bar{d} + 1.0)^2 - 1} \right),$$

which is roughly $2 - \frac{6.28}{\bar{d}+6.74}$ by using $r = 1.0741$ in [7].

Figure 1 compares approximation factors of the lower envelope of VCPM-Apx(w), the Chen and Kanj's algorithm, and the Halldórsson and Radhakrishnan's. VCPM-Apx(w) outperforms theirs for all \bar{d} . We note that the Halldórsson and Radhakrishnan's algorithm has an approximation factor of $2 - \frac{5}{2\bar{d}+3}$ for any graph such that $\text{Opt}(G) \geq \frac{|V|}{2}$ (not only the graphs with perfect matchings).

Remark. In [4], an interesting reduction is shown. It reduces an r -approximation algorithm for VC-PM to a $\max\{1.5, (2r+2)/3\}$ -approximation algorithm for general VERTEX COVER. Unfortunately, however, the reduction of neither their algorithm nor ours derives improvement on approximation factor of the general VERTEX COVER, since their reduction does not preserve the average degree.

A graph $G = (V, E)$ is called everywhere k -sparse if $\forall V' \subseteq V \ |E(G(V'))| \leq k|V'|$. Note that everywhere k -sparse graphs have average degree $k/2$. It is also shown in [4] that for everywhere k -sparse graph the average degree is preserved by the reduction and our algorithm can find an approximate solution of factor $\max\{1.5, 2 - \frac{4.18}{2k+6.74}\}$.

5 For Graphs with Large Matchings

In this section we analyze $VCPM\text{-}Ap\mathbf{x}(w)$ when they are applied to general graphs. We show that it performs well for graphs that have “large” matchings, although its approximation factor is degenerated.

Let $G = (V, E)$ be an instance graph, where $n = |V|$ and $m = |E|$. Let M be one of the maximum matchings of G which is not necessarily perfect, and let pn be the number of vertices in M , where $0 \leq p \leq 1$ is the ratio of the size of M over the size of perfect matching. Given such a matching M , we construct the formula F_G as in section 3.

In the following we show lemmas 7, 8, 9 and theorem 2, extending lemmas 2, 3, 4 and theorem 1, respectively. We show the proof of only Lemma 8 and omit the others since they are slight modifications of the original ones.

Lemma 7. $|C| \leq n + \frac{1}{w}(m - |\sigma_C|)$

Lemma 8. $Opt(G) \geq pn + \frac{1}{w}(m - Opt(F_G))$

Proof. We separate V into the following four sets: $C_0 = V(M) \cap C$, $I_0 = V(M) \cap \overline{C}$, $C_1 = \overline{V(M)} \cap C$ and $I_1 = \overline{V(M)} \cap \overline{C}$. It is clear that $Opt(G) = |C| = |C_0| + |C_1| = n - |I_0| - |I_1|$.

Since C is a vertex cover of G , σ_C satisfies all positive clauses of F_G . On the other hand, if a negative clause $(\overline{x}_u \vee \overline{x}_v)$ is satisfied by σ_C then one of the variables in $\{x_u, x_v\}$ is assigned to TRUE and the other to FALSE. Therefore, the number of negative clauses satisfied by σ_C is equal to $|I_0|$. It follows that $|\sigma_C| = m + w|I_0|$. In addition $Opt(F_G) \geq |\sigma_C|$, so we obtain $Opt(F_G) \geq m + w|I_0|$. Then, the following inequality holds.

$$\begin{aligned} Opt(F_G) &\geq m + w|I_0| \\ &= m + w(n - Opt(G) - |I_1|) \\ &\geq m + w(pn - Opt(G)) \end{aligned}$$

The lemma follows by expelling $Opt(G)$ from it.

Lemma 9.

$$\frac{|C|}{Opt(G)} \leq \frac{2}{p} - 1 + \frac{2}{pw}\delta + \frac{r-1}{wr} \left(\frac{1}{p}\overline{d}(G) + w \right)$$

These lemmas provide the following approximation factor of $VCPM\text{-}Ap\mathbf{x}(w)$ for graphs with large matchings.

Theorem 2. $VCPM\text{-}Ap\mathbf{x}(w)$ achieves an approximation factor

$$\frac{2}{p} - 2 \frac{\frac{1-r}{p}\overline{d} + w}{rw(w+1)}$$

for graphs of average degree \overline{d} and having a matching of size at least $pn/2$.

Halldórsson and Radhakrishnan's algorithm applied to graphs with large matchings achieves a factor $\frac{2}{p} - (\frac{2}{p} - 1) \frac{5}{2d+3}$. For large p , VCPM-Apx(w) is better than theirs. For example, when $p = 0.95$ our algorithm outperforms theirs when $\bar{d} \geq 1.528$, and reaches to 2 when $\bar{d} = 50$. When $p \leq 0.825$, on the other hand, our algorithm becomes worse than theirs for all \bar{d} , although in this case both can have factors of less than 2 only when $\bar{d} \leq 7$.

References

1. Brenda S. Baker. Approximation algorithms for np-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.
2. R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–45, 1985.
3. P. Berman and T. Fujito. On approximation properties of the independent set problem in degree 3 graphs. *Lecture Notes in Computer Science*, 955:449–460, 1995.
4. Jianer Chen and Iyad Kanj. On approximating minimum vertex cover for graphs with perfect matching. In *International Symposium on Algorithms and Computation*, pages 132–143, 2000.
5. Andrea E. F. Clementi and Luca Trevisan. Improved non-approximability results for minimum vertex cover with density constraints. *TCS: Theoretical Computer Science*, 225(1-2):113–128, 1999.
6. Irit Dinur and Shmuel Safra. The importance of being biased. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC-02)*, pages 33–42, New York, May 19–21 2002. ACM Press.
7. Uriel Feige and Michel X. Goemans. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *Proceedings of the Third Israel Symposium on Theory of Computing and Systems*, 1995.
8. M. Garey and D. Johnson. Computers and intractability: A guide to the theory of npcompleteness w, 1979.
9. Magnus M. Halldórsson and Jaikumar Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18:145–163, 1997.
10. Eran Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM Journal on Computing*, 31(5):1608–1623, October 2002.
11. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 6, 1983.
12. Marek Karpinski and Alexander Zelikovsky. Approximating dense cases of covering problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 4(004), 1997.
13. S. Khot and O. Regev. Vertex cover might be hard to approximate to within $2 - \epsilon$. In *Proc. of 18th IEEE Annual Conference on Computational Complexity (CCC)*, pages 379–386, 2003.
14. Burkhard Monien and Ewald Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica*, 22(1):115–123, April 1985. NCPY.

An Approximation Algorithm for Weighted Weak Vertex Cover Problem in Undirected Graphs^{*}

Yong Zhang and Hong Zhu

Department of Computer Science and Engineering, Fudan University, China
Laboratory for Intelligent Information Processing, Fudan University, China
{012021133,hzhu}@fudan.edu.cn

Abstract. The problem of efficiently monitoring the network flow is regarded as the one to find out the minimum weighted weak vertex cover set for a given graph $G = (V, E)$ with weight function w . In this paper, we show that a weak vertex cover set approximating a minimum one within $2 - \frac{2}{\nu(G)}$ can be efficiently found in undirected graphs, and improve the previous work of approximation ratio within $\ln d + 1$, where d is the maximum degree of the vertex in graph G .

1 Introduction

As internet growing rapidly, more and more network applications need to know the network flow information to monitor network utilization and performance. Most monitoring technologies typically assume that the measurement instrumentation can be either intelligently distributed at different points in the underlying network[1] or placed at the endpoints of the end-to-end path whose characteristics are of interest[2]. And now, more and more data must be collected by network monitoring process at much higher frequencies. Then the overhead that monitoring method imposes on the underlying router can be significant and adversely impact the router's throughput and severely impact its performance.

We describe the problem of efficiently monitoring the network flow as finding the minimum **Flow Monitoring Set** of a graph G . Formally,

Definition 1 [3](Flow Monitoring Set) *Given an undirected graph $G=(V,E)$, where V denotes the set of nodes, E represents the edges between two nodes, we say $S \subseteq V$ is a flow monitoring set of G , if monitoring the flow of those edges that are incident on nodes in S is sufficient to infer the flow of every edge in E . And the following two constraints must be satisfied:*

- (1) $\forall v \in V$, $\text{Degree}(v) \geq 2$, where $\text{Degree}(v)$ denotes the degree of node v
- (2) $\forall v \in V$. $\sum_{u \in V} f(u, v) = 0$, where $f(u, v)$ denote the flow from node u to node v .

^{*} This work is supported by a grant from the Ministry of Science and Technology (grant no. 2001CCA03000), National Natural Science Fund (grant no. 60273045) and Shanghai Science and Technology Development Fund (grant no. 03JC14014)

The minimum flow monitoring set is a flow monitoring set which contains minimum number of nodes of the graph. When the nodes of the graph are of non-negative weights, we call it the problem of weighted minimum flow monitoring set of graph G .

Constraint (2) shows that the flow monitoring set problem satisfies flow conservation. So, we can demonstrate that the problem of finding the minimum flow monitoring set from an underlying network gives rise to the problem of finding the minimum Weak Vertex Cover set[5] of a graph. In general, Weak Vertex Cover is a Vertex Cover Problem[11] enriched with a linear system of equations for edge variables representing additional "knowledge" that can be used to reduce the size of cover set. Hence we replace solving the problem of finding the minimum flow monitoring set by solving the problem of finding the minimum **weak vertex cover set** for a given graph.

Definition 2 (*Weak Vertex Cover*) *Given an undirected graph $G = (V, E)$, where $\forall v \in V, \deg(v) \geq 2$ holds, we say $S \subseteq V$ is a Weak Vertex Cover Set of G , if and only if every edge in G can be marked by performing the following three steps:*

- (1) *Mark all edges that are incident on vertices in S .*
- (2) *Mark the edge if it is the only unmarked edge among all of the edges which are incident on the same vertex.*
- (3) *Repeat step (2) until no new edge can be marked.*

We call it **the weighted weak vertex cover problem** when each vertex v has weight w_v associated with it. Hence the weighted weak vertex cover problem is to identify the smallest weight collection of vertices, and the vertex collection is a weak vertex cover set.

For solving the problem of finding the minimum weak vertex cover set, [3] proves that $S \subseteq V$ is a weak vertex cover set of undirected graph $G = (V, E)$ iff $G' = (V', E')$ is a forest, where $V' = V - S$ and $E' = \{(u, v) \mid (u, v) \in E \cap u \in V' \cap v \in V'\}$, then brings forward a greedy approximation algorithm which gives an approximation ratio $2(\ln d + 1)$, where $d = \max_{v \in V} \{\deg(v)\}$. And [4] proves that the weak vertex cover problem is \mathcal{NP} -hard, then gives an approximation algorithm with approximation ratio $\ln d + 1$. Algorithm in [4] based on some property of cycle space in graph theory[12], and some of those properties are still used in this paper.

The paper is structured as follows. The problem is brought forward, and some basic definition and notation used throughout the paper are given in the Introduction section. We show a well character for cycle-rank-proportional graph in the next section, which provides a foundation for our algorithm. And in section 3, we give an approximation algorithm to solve the minimum weighted weak vertex cover problem with approximation ratio of $2 - \frac{2}{\nu(G)}$, where $\nu(G) = |E| - |V| + 1$. Then we bring forward our further research in the last section.

2 Well Character for Cycle-Rank-Proportional Graph

Definition 3 In a weighted graph $G = (V, E)$, a weight function w is **Cycle-Rank-Proportional** if for some constant $c > 0$, we have $w(u) = c \cdot R_u$ for every $u \in V$, where R_u is the number of decreased cycle rank of cycle space after deleting vertex u .

We call the dimension of cycle space of graph $G = (V, E)$ the *cyclomatic number*[12] of G , namely, $\nu(G) = |E| - |V| + p_G$, where p_G is the number of connected components in G . Hence,

$$\begin{aligned} R_u &= \nu(G) - \nu(G - \{u\}) \\ &= (|E| - |V| + p_G) - ((|E| - d(u)) - (|V| - 1) + p_{G-\{u\}}) \\ &= d(u) - k_u \end{aligned}$$

where k_u is the increased number of connected components plus 1 after deleting u .

fact 1 In Cycle-Rank-Proportional Graph, $w(u) = c \cdot (d(u) - k_u)$ for every $u \in V$.

Definition 4 A cycle $C = \{v_1, \dots, v_k\}$ is a **nearly-simple-cycle**, if it has at most one vertex v_i , ($1 \leq i \leq k$), which incidents with other vertices not in the cycle.

In this section, the weight function in graph we discussed is *Cycle-Rank-Proportional*, even more, we can look it as $w(u) = d(u) - k_u$. And the graph we discussed in this section is connected and without nearly-simple-cycle.

Lemma 1 For an arbitrary Weak Vertex Cover Set S of a connected graph $G = (V, E)$, $w(S) \geq \nu(G)$.

Proof: After deleting all the vertices in S and edges incident with them, no cycle remains in the graph. The rank of cycle space in graph G is $|E| - |V| + t$, where t is the number of connected components in G , in a connected graph, $t = 1$. Hence,

$$w(S) = \sum_{v \in S} w(v) \geq |E| - |V| + 1 = \nu(G)$$

■

Lemma 2 If S is a minimal Weak Vertex Cover Set in a connected graph $G = (V, E)$, where G contains no nearly-simple-cycle, then $w(S) \leq 2 \cdot (\nu(G) - 1)$.

Proof: After deleting each vertex u in S , it will produce k_u ($k_u \geq 1$) components.

Since S is a weak vertex cover set in graph G , after deleting S and edges incident with vertices in S , the remained graph is a forest, say F , and $F = \{T_i\}$. Let $\delta(T_i)$ denote the number of edges who connect tree T_i and S , where $T_i \in F$.

Because S is a minimal weak vertex cover set, any vertex in S has at least two edges connect to some tree in F . We call the collection of such vertices S_T that each of them belongs to S and has at least two edges incident with tree T .

Let p be a potential function in V , s.t. $p(u) = d(u) - 2$ for every $u \in V$. And T be an arbitrary tree in F , assume that T has t vertices, then in this subgraph,

$$p(T) = \sum_{u \in T} (d(u) - 2) = \sum_{u \in T} d(u) - 2t = \delta(T) + 2(t - 1) - 2t = \delta(T) - 2$$

In this section, we only discuss the cycle-rank-proportional graphs, hence, any vertex in graphs is contained in one cycle at least. Therefore, for an arbitrary tree T in F , $\delta(T) \geq 2$. In case $S_T \neq \emptyset$, if $\delta(T) = 2$, a nearly-simple-cycle exist, we will analysis it in the following section. Since $S_T \leq \lfloor \frac{1}{2}\delta(T) \rfloor$, each vertex in S_T can receive an extra potential of $\frac{\delta(T)-2}{\lfloor \frac{1}{2}\delta(T) \rfloor}$. It is at least 1 if $\delta(T) \geq 3$.

Since any vertex in S must in some S_T , and remember the fact that $k_u \geq 1$, we can get:

$$w(S) = \sum_{u \in S} (d(u) - k_u) \leq \sum_{u \in S} (d(u) - 1) \leq \sum_{u \in V} (d(u) - 2) = 2|E| - 2|V|$$

As the cyclomatic number $\nu(G) = |E| - |V| + 1$, therefore:

$$w(S) \leq 2 \cdot (\nu(G) - 1)$$

■

From the above two lemma, we can easily draw our conclusion as the following one.

Theorem 1 *S is any minimal Weak Vertex Cover Set of Cycle-Rank-Proportional graph G without nearly-simple-cycle, w is its weight function and $\text{opt}(G, w)$ is the minimum Weak Vertex Cover Set in graph G , then:*

$$w(S) \leq (2 - \frac{2}{\nu(G)}) \cdot w(\text{opt}(G, w))$$

3 Approximation Algorithm for Weighted Weak Vertex Cover

For \mathcal{NP} -hard optimization problems[11], we often design a polynomial time approximation algorithm[6, 7] to get the solution which is close to the optimum solution. We call an algorithm for a minimum optimization problem has an **approximation ratio** of $\rho(n)$ if, for any input of size n , the cost C of the solution produced by the algorithm is within a factor of $\rho(n)$ of the cost C^* of an optimum solution: $\max(\frac{C}{C^*}) \leq \rho(n)$ [8].

In the first part of this section, We will first introduce the Local-Ratio Theorem[10], which was pioneered by Bar Yehuda and Even, who used it to develop an approximate graph vertex cover algorithm. Then we give our algorithm for weighted weak vertex cover problem by applying this theorem.

3.1 Introduction to Local-Ratio Theorem

Theorem 2 [9] (*Local-Ratio Scheme*) *If a feasible solution is r -approximation w.r.t a pair of weight functions $W1$ and $W2$ then it is also an r -approximation w.r.t $W1+W2$.*

We introduce the definition of decomposition before formalizing the above theorem to the following one, hence we can apply it to our algorithm directly.

Definition 5 *For a weighted undirected graph $G = (V, E)$, w is its weight function, a set of weight functions $\{w_i : i = 1, \dots, k\}$ defined on V is a **decomposition** of w if $\sum_i w_i(u) \leq w(u)$ for each $u \in V$.*

Theorem 3 [10] (*Local Ratio Theorem*) *In a weighted graph $G = (V, E)$, w is its weight function, let $\{w_i, i = 1, \dots, k\}$ be a decomposition of w , and S be any Weak Vertex Cover Set of G s.t. $w(S) = \sum_i w_i(S)$. Then,*

$$\frac{w(S)}{w(\text{opt}(G, w))} \leq \max_i \left\{ \frac{w_i(S)}{w_i(\text{opt}(G, w_i))} \right\}$$

where $\text{opt}(G, w)$ denotes the optimum Weak Vertex Cover Set of G in weight function w .

Proof: Since w_i is a decomposition of w , for any set $U \subseteq V$, $w(U) \geq \sum_i w_i(U)$. Besides, $G(w_i)$ is a subgraph of $G(w)$ for every i , and if S is a Weak Vertex Cover Set of graph G , then its restriction to any graph G' , where $G' \subseteq G$, is still a Weak Vertex Cover Set of G' . Therefore, we have $w_i(\text{opt}(G, w)) \geq w_i(\text{opt}(G, w_i))$ for all i . It follows now that

$$\begin{aligned} \frac{w(S)}{w(\text{opt}(G, w))} &\leq \frac{\sum_i w_i(S)}{\sum_i w_i(\text{opt}(G, w))} \leq \frac{\sum_i w_i(S)}{\sum_i w_i(\text{opt}(G, w_i))} \\ &\leq \max_i \left\{ \frac{w_i(S)}{w_i(\text{opt}(G, w_i))} \right\} \end{aligned}$$

■

3.2 Description and Analysis of Algorithm

From Local-Ratio Theorem, we can decompose the weight function w of G into cycle-rank-proportional weight functions. And since the weight of any minimal weak vertex cover set is within $2 - \frac{2}{\nu(G)}$ of the optimum in cycle-rank-proportional graph, therefore, an approximation algorithm for weight weak vertex cover problem with approximation of $2 - \frac{2}{\nu(G)}$ can be designed.

The following algorithm *WVC* contains two main loops. The first loop decompose the weight function w into $\{w^j\}$, where every w^j are cycle-rank-proportional for graph G . The second loop delete the redundant vertices step by step, hence, the remained vertices in any stage are minimal weak vertex cover set.

$WVC(G, w)$

1. find the number k of connected components of graph G
2. $\nu(G) = |E| - |V| + k$; $S = \phi$
3. $j = 0$; $w_j = w$; $s = \nu(G)$
4. **while** $V(w_j) \neq \phi$ and $s > 0$
 - // $V(w_j)$ denotes the largest subset of V ,
 - // each vertex u in it satisfying $w_j(u) > 0$
5. **if** a nearly-simple-cycle (v_1, \dots, v_k) contained in graph $G(V(w_j))$
6. **then**
7. select the vertex v_i whose weight is minimal in $\{v_1, \dots, v_k\}$
8. $w^j(v) = \begin{cases} w_j(v_i) & \text{if } v \in \{v_1, \dots, v_k\}, \\ 0 & \text{otherwise.} \end{cases}$
9. **else**
10. $\Delta = \min\{\frac{w_j(v)}{d(v, G(V(w_j))) - k_v} \mid v \in V(w_j)\}$
 - // $d(v, G(V(w_j)))$ denotes the degree of v in graph $G(V(w_j))$
11. $w^j(v) = \Delta \cdot (d(v, G(V(w_j))) - k_v)$ if $v \in V(w_j)$
12. $w_{j+1} = w_j - w^j$
13. $S_j = V(w_j) - V(w_{j+1})$
14. find the number k_{j+1} of connected components of graph $G(V(w_{j+1}))$
15. $s = |E(w_{j+1})| - |V(w_{j+1})| + k_{j+1}$
16. $j++$
17. **for** $(t = j - 1 ; t \geq 0 ; t--)$
18. $S = S \cup S_t$
19. **for** every $u \in S_t$ **do**
20. **if** $V(w_t) - (S - u)$ induces no cycle in $G(V(w_t))$
21. **then** remove u from S

fact 2 $w(u) \geq \sum_j w^j(u)$ for every $u \in V$.

fact 3 Output S of algorithm WVC satisfies: $w(S) = \sum_j w^j(S)$

From the above two fact and Local-Ratio-Theorem, we can get our conclusion, formally in the following theorem:

Theorem 4 The approximation of algorithm WVC is $2 - \frac{2}{\nu(G)}$.

Proof: Each subgraph G' according to weight function w^j , which belongs to the decomposition of weight function w , must either a nearly-simple-cycle, or a cycle-rank-proportional graph. Analysis in section 2 shows that the total weight of minimal weak vertex cover set of a cycle-rank-proportional graph is within $2 - \frac{2}{\nu(G)}$ of the optimum. In the nearly-simple-cycle, the vertex we chosen in the algorithm is of the same weight with the optimum of the weighted subgraph.

Therefore, the total weight of the algorithm's output is within $2 - \frac{2}{\nu(G)}$ of the optimum. ■

Theorem 5 *The running time of algorithm WVC is*

$$\mathcal{O}(|V| \cdot (|E| + |V|) \cdot \min\{\nu(G), |V|\})$$

Proof: Line 1 can be finished in $\mathcal{O}(|E| + |V|)$ by \mathcal{BFS} algorithm. Line 2 and line 3 run in constant time. Line 5 takes $\mathcal{O}(|V|)$ since it maybe check all the vertex. Since w is a vector of rank $|V|$, line 6 to line 8 takes $\mathcal{O}(|V|)$ time. Compute k_v takes $\mathcal{O}(|E| + |V|)$ by \mathcal{BFS} algorithm, hence, line 11 can be finished in $\mathcal{O}(|V| \cdot (|E| + |V|))$ times. And line 12 to line 16 takes $\mathcal{O}(|V| + |E|)$ time. Notice that the while loop works at most $\min\{\nu(G), |V|\}$ times. Hence, the total running time of the while loop is $\mathcal{O}(|V| \cdot (|E| + |V|) \cdot \min\{\nu(G), |V|\})$.

Since $j \leq \min\{\nu(G), |V|\}$, and line 18 to line 21 takes $\mathcal{O}(|V|)$, the total running time of the for loop is $\mathcal{O}(|V| \cdot \min\{\nu(G), |V|\})$.

Therefore, the total time of algorithm WVC is

$$\mathcal{O}(|V| \cdot (|E| + |V|) \cdot \min\{\nu(G), |V|\})$$

■

4 Further Research

In the paper, we give an approximation algorithm for the minimum weighted weak vertex cover problem, whose approximation ratio is $2 - \frac{2}{\nu(G)}$, improve the previous work. Can we improve this approximation ratio into a better one? Or can we prove the inapproximation of this problem? Our research in the future will focus on these questions. And further, since the constraint of flow conservation equation only holds approximately, our research will focus on estimating the influence of the approximation error on the flow of edges calculated by this method, and try to give an algorithm solving the problem with approximation error.

References

1. Jamin S, Jin C, Jin Y, Raz D, Shavitt Y, Zhang L, On the placement of internet instrumentation. In Proceedings of the IEEE INFOCOM 2000. Tel-Aviv, Israel. 2000.26-30
2. Lai K, Baker M, Measuring bandwidth. In Proceedings of the IEEE INFOCOM'99. New York, 1999.235-245
3. Liu Xianghui, et al. Analysis of Efficient Monitoring Method for the Network Flow. Journal of Software, 2003,14(2):300-304
4. Zhang Yong and Zhu Hong, An Approximation Algorithm for Weighted Weak Vertex Cover. Journal of Computer Science and Technology, accepted.
5. Breibart Y, Chan CY, Carofalakis M, Rastogi R, Silberschatz A, Efficiently monitoring bandwidth and latency in IP network. In Proceedings of the IEEE INFOCOM 2001. Alaska, USA. 2001.933-942.
6. Dorit S. Hochbaum, Approximation Algorithm for \mathcal{NP} -Hard Problems, PWS Publishing Company, 1997.

7. Vijay V.Vazirani, Approximation Algorithms. Springer-Verlag, 2001.
8. T H. Corman, C E. Leiserson, R L. Rivest, C Stein, Introduction to Algorithms, Second Edition, The MIT Press, 2001.
9. R. Bar-Yehuda and S. Even, A local-ratio theorem for approximating the weighted vertex cover problem. Ann. Discrete Math., 25:27–45, 1985.
10. Vineet Bafna, Piotr Berman, Toshihiro Fujito, Constant Ratio Approximations of the Weighted Feedback Vertex Set Problem for Undirected Graphs. ISAAC 1995: 142-151
11. M.R.Garey and D.S.Johnson, Computers and intractability: A Guide to the Theory of NP-Completeness. W.H.Freeman, 1979.
12. Michel Gondran and Michel Minoux, Graph and Algorithms, John Wiley & Sons Publication, March 1984.

On the Arrangement of Cliques in Chordal Graphs with Respect to the Cuts

L. Sunil Chandran¹ and N.S. Narayanaswamy²

¹ Max-Planck Institute for Informatik,
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
Phone: +49 681 9325504, Fax: +49 681 9325599

sunil@mpi-sb.mpg.de

² Department of Computer Science and Engineering, Indian Institute of Technology,
Chennai-600 036, India
swamy@shiva.iitm.ernet.in

Abstract. A cut (A, B) (where $B = V - A$) in a graph $G(V, E)$ is called internal, iff there exists a node x in A which is not adjacent to any node in B and there exists a node $y \in B$ such that it is not adjacent to any node in A . In this paper, we present a theorem regarding the arrangement of cliques in a chordal graph with respect to its internal cuts. Our main result is that given any internal cut (A, B) in a chordal graph G , there exists a clique with $\kappa(G) + 1$ nodes (where $\kappa(G)$ is the vertex connectivity of G) such that it is (approximately) bisected by the cut (A, B) . In fact we give a stronger result: For any internal cut (A, B) of a chordal graph, for each i , $0 \leq i \leq \kappa(G) + 1$, there exists a clique K_i such that $|K_i| = \kappa(G) + 1$, $|A \cap K_i| = i$ and $|B \cap K_i| = \kappa(G) + 1 - i$.

An immediate corollary of the above result is that the number of edges in any internal cut (of a chordal graph) should be $\Omega(k^2)$, where $\kappa(G) = k$. Prompted by this observation, we investigate the size of internal cuts in terms of the vertex connectivity of the chordal graphs. As a corollary, we show that in chordal graphs, if the edge connectivity is strictly less than the minimum degree, then the size of the mincut is at least $\frac{\kappa(G)(\kappa(G)+1)}{2}$, where $\kappa(G)$ denotes the vertex connectivity. In contrast, in a general graph the size of the mincut can be equal to $\kappa(G)$. This result is tight.

1 Introduction

Let C be a cycle in a graph G . A chord of C is an edge of G joining two vertices of C which are not consecutive. A graph G is called a chordal graph iff every cycle in G of length 4 or more has a chord. Chordal graphs arise in many applications (see [7, 10, 15]), which includes the study of evolutionary trees [1], facility location [5], scheduling problems [11], and the solution of sparse systems of linear equations [12, 13]. Chordal graphs constitute one of the most important subclasses of perfect graphs [7]. The various names given to chordal graphs, such as triangulations, rigid circuit graphs, perfect elimination graphs, monotone transitive graphs and so on illustrate the importance of these graphs.

Let $G = (V, E)$ be a connected undirected graph. Throughout this paper we will use V for the set of vertices of G and E for the set of edges. $|V|$ will be denoted by n . $N(v)$ will denote the set of neighbours of v , that is $N(v) = \{u \in V : (u, v) \in E\}$. For $A \subseteq V$, we use $N(A)$ to denote the set $\bigcup_{v \in A} N(v) - A$. The subgraph of G induced by the nodes in A will be denoted by $G(A)$.

The *vertex connectivity* $\kappa(G)$ of an undirected graph is defined to be the minimum number of vertices whose removal results in a disconnected graph or a trivial graph (ie, single node). A subset $S \subset V$ is called a *separator* if $G(V - S)$ has at least two components. Then, $\kappa(G)$ is the size of the minimum sized separator. Note that a complete graph has no separator and its (vertex) connectivity is by definition $n - 1$.

A cut $(A, V - A)$ where $\emptyset \neq A \subset V$ is defined as the set of edges with exactly one end point in A . That is, $(A, V - A) = \{(u, v) \in E : u \in A \text{ and } v \in V - A\}$. In this paper we always use (A, B) instead of $(A, V - A)$ for notational ease, i.e., B always means $V - A$.

Definition 1. A node $x \in A$ is called a *hidden node with respect to B* (or with respect to the cut (A, B)), iff $x \notin N(B)$.

Definition 2. A cut (A, B) is called an *internal cut of G* iff there exists a hidden node $x \in A$ and a hidden node $y \in B$. Equivalently, a cut (A, B) is internal iff $N(A) \neq B$ and $N(B) \neq A$, ie, $N(A)$ is a proper subset of B and $N(B)$ is a proper subset of A .

Note that if x is a hidden node in A , then $\{x\} \cup N(x) \subseteq A$. In fact, the only condition for (A, B) to be an internal cut is that there should exist a node $x \in A$ and $y \in B$ such that $N(x) \subseteq A$ and $N(y) \subseteq B$. Thus if we can find two non adjacent nodes x and y , such that $N(x) \cap N(y) = \emptyset$, then there exist internal cuts in the graph: we just have to include $\{x\} \cup N(x)$ in A and $\{y\} \cup N(y)$ in B and divide the remaining nodes arbitrarily. The reader can easily verify that this can be done in $2^{n - |\{x\} \cup N(x)| - |\{y\} \cup N(y)|}$ ways. Thus in general, the number of internal cuts in a graph can be exponential.

Our results: We present a structural theorem about internal cuts in chordal graphs. Let (A, B) be an internal cut in a chordal graph G with connectivity $\kappa(G) = k$. Let i be a number such that $0 \leq i \leq k + 1$. Then we prove that there exists a clique K_i in G , such that $|K_i| = k + 1$, $|K_i \cap A| = i$ and $|K_i \cap B| = k + 1 - i$. (Theorem 3).

An interesting special case of the above result is when $i = \lfloor \frac{k+1}{2} \rfloor$. Then, the theorem states that, irrespective of which internal cut (A, B) we consider, it is always possible to find a clique of size $k + 1$, such that it is bisected by (A, B) . We came up with this problem while trying to prove a theorem about the *treewidth* of a certain class of graphs which happened to have reasonably high connectivity. Note that for any graph G , $\text{treewidth}(G) + 1$ is the same as the minimum possible clique number (i.e., the number of nodes in the maximum clique) over all possible chordal graphs G' such that G' is a super graph of G . In order to get the desired result, the approach we followed required that we should

be able to find some cut (A, B) in the chordal completion of G , such that $|A|$ and $|B|$ are more or less of the same size. Moreover, we wanted to ensure that no “big cliques” in the graph is bisected by the cut, or at least that given any big clique, the majority of the nodes in this clique belonged to one of the sides. But irrespective of however we increased the assumptions on the structure of the graph classes we were studying, we were unable to find nice chordal completions with such nice cuts. Working with small example graphs, we realised that when we try to “adjust” a cut (by moving certain nodes from A to B and so on) such that a certain clique is not bisected, invariably some other clique gets bisected. This led us to exploring the arrangement of cliques with respect to cuts in chordal graph.

As of now, we are unable to present any specific applications for our theorem. But we would like to point out that the chordal completions of graphs are of great interest from the point of view of two important optimisation problems: the treewidth and the fill-in problem. As mentioned in the previous paragraph, given a graph G , the treewidth problem is to find a chordal completion of G , so as to minimise the clique number. The fill-in problem seeks to find a chordal completion of G , which minimises the number of edges added. Also note that, properties such as connectivity are monotonic: i.e., as new edges are added, the property never decreases. Thus, the value of the property in the chordal completion will be at least as much as in the original graph. Structural understanding of chordal graphs, with respect to such properties turn out to be useful, in making apriori inferences about the optimal chordal completions. (For example, see [4], where lower bounds for treewidth and pathwidth are derived in terms of a generalised connectivity property.)

Moreover, chordal graphs are well-studied from various perspectives. For example, the vertex separators in chordal graphs are well understood. But the corresponding concept of edge separator, namely the cuts (due to some reason) are very little studied. In this paper, we attempt to understand chordal graphs, from the perspective of edge cuts.

As mentioned before, the number of internal cuts can be exponential, in general. Then does the fact that every internal cut “cuts through” many cliques of size $k + 1$ mean that always there exist a large number of cliques in a chordal graph? Not necessarily. It is well known that the number of maximal cliques (which can sometimes contain as small as $k + 1$ nodes) in a chordal graph is upper bounded by n . (See [7].) It is just that these few cliques, arrange themselves in such a way, that it is possible for every internal cut to “cut through” a few of them. Thus the result indeed throws some light on the arrangement of cliques with respect to cuts in a chordal graph.

Another result inspired by the above theorem: An immediate consequence of the result (Theorem 3) is that the size of every internal cut has to be $\Omega(\kappa(G)^2)$. This is because for $i = \lfloor \frac{\kappa(G)+1}{2} \rfloor$ the internal cut would be approximately “bisecting” a clique K_i with $(\kappa(G) + 1)$ nodes. After observing this, we investigated whether a better result is possible. In Theorem 4 we prove a lower bound for the size of internal cuts in chordal graphs, namely $\frac{\kappa(G)(\kappa(G)+1)}{2}$, which is better than

what can be immediately inferred from Theorem 3. We also show that this result is strict in the sense that there exist chordal graphs with vertex connectivity $\kappa(G)$ and having internal cuts with exactly $\frac{\kappa(G)(\kappa(G)+1)}{2}$ edges.

The *edge connectivity* $\lambda(G)$ of a graph G is defined to be the minimum number of edges whose removal results in a disconnected graph or a trivial graph. A *minimum cut* is a cut consisting of $\lambda(G)$ edges.

It is well known that the following inequality (due to Whitney [14]) holds in a general undirected graph:

$$\kappa(G) \leq \lambda(G) \leq \delta(G)$$

where $\delta(G)$ is the minimum degree of the graph. For a discussion of this inequality and for related work see the chapter on connectivity in [8]. In [6], Chartrand and Harary show that for all integers a, b, c , such that $0 < a \leq b \leq c$, there exists a graph G with $\kappa(G) = a$, $\lambda(G) = b$ and $\delta(G) = c$.

We study this inequality when restricted to the class of chordal graphs. We observe that when $\lambda(G) < \delta(G)$, every mincut in G is also an internal cut. Thus, as a corollary of our result on internal cuts in chordal graphs, we show that there is a “gap” between $\lambda(G)$ and $\kappa(G)$ in chordal graphs provided $\lambda(G) \neq \delta(G)$. More specifically, if $\lambda(G) \neq \delta(G)$, then $\lambda(G) \geq \frac{\kappa(G)(\kappa(G)+1)}{2}$. This improves the result in [2] which shows that $\lambda(G) \geq 2\kappa(G) - 1$, when $\lambda(G) < \delta(G)$. And the lower bound we obtain here is tight.

1.1 Preliminaries

The following result called *Menger’s theorem* is well known among graph theorists. An elegant proof due to Dirac is given in [8].

Theorem 1. [8]**Menger’s theorem:** *The minimum number of vertices separating two nonadjacent nodes s and t is equal to the maximum number of vertex disjoint $s - t$ paths.*

Thus, there are at least $\kappa(G)$ vertex disjoint paths between every pair of nonadjacent nodes of G .

A bijection $f : V \rightarrow \{1, 2, \dots, n\}$ is called an ordering of the vertices of G . Then $f(v)$ is referred to as the number associated with the vertex v , or simply the *number of v* with respect to the ordering f . Given an ordering f of a graph G , we define the following terms.

Definition 3. Let $A \subseteq V$. The *highest*(A) is defined to be the vertex with the highest number in A .

Definition 4. A path $P = (w_1, w_2, \dots, w_k)$ in G is called an *increasing path*, iff $f(w_1) < f(w_2) < \dots < f(w_k)$. It is called a *decreasing path* iff $f(w_1) > f(w_2) > \dots > f(w_k)$. A single node can be considered as either increasing or decreasing.

Definition 5. A vertex $u \in N(v)$ is called a *higher neighbour* of v iff $f(u) > f(v)$. The set of higher neighbours of v will be denoted by $N_h(v)$, ie

$$N_h(v) = \{u \in N(v) : f(u) > f(v)\}$$

Definition 6. An ordering f of G is called a perfect elimination ordering (PEO) iff for each $v \in V$, $G(\{v\} \cup N_h(v))$ is a complete subgraph (clique) of G . Then $f(v)$ will be called the PEO number of v .

Note that every graph may not have a PEO. The following theorem is well known (see [7]).

Theorem 2. [7] An undirected graph G is chordal if and only if there exists a PEO for G .

A node v is called *simplicial* iff $N(v)$ induces a clique. The following observation is due to Dirac:

Lemma 1. [7] In every chordal graph, which is not a complete graph, there exists at least two non adjacent simplicial nodes. (In a complete graph, every node is simplicial).

From Lemma 1, it is easy to make the following useful observation. (See for example, [7], page 84).

Lemma 2. Suppose v is any node in a chordal graph G . Then there exists a PEO f for G such that $f(v) = n$.

In fact more can be inferred. (See also, [9], page 8). Let K be a clique in a chordal graph G . Then from Lemma 1, it can be inferred that if $V(G) - K \neq \emptyset$, there should be a simplicial vertex $x_1 \in V(G) - K$. Let $f(x_1) = 1$. Now remove x_1 from G . Clearly, the induced subgraph on the remaining nodes (say G') is also chordal and thus Lemma 1 is applicable to G' also. Then we can find a simplicial vertex $x_2 \in V(G') - K$, which is simplicial (as long as $V(G') - K$ is nonempty). Let $f(x_2) = 2$. By repeating this procedure, one can get a PEO f for G such that the nodes of K gets the highest numbers, namely the numbers from $n - |K| + 1$ to n . We state this useful observation as a lemma.

Lemma 3. Let K be any clique in a chordal graph G . Then there exists a PEO f of G such that the highest $|K|$ numbers are assigned to the nodes of K . In other words, $n - |K| + 1 \leq f(x) \leq n$, for each node $x \in K$.

Another well known characterisation of chordal graphs is in terms of minimal separators. (A separator S of two nodes x and y is said to be a minimal separator if no proper subset of S can separate x from y . Note that every separator of x and y should contain a minimal separator of x and y).

Lemma 4. A graph G is a chordal graph if and only if every minimal separator of G induces a complete subgraph.

A chordless path from u to v is defined to be a path from u to v in G such that no two non consecutive nodes of the path are adjacent. The reader can easily verify that if there is a path between u and v then there is a chordless path also. For example, the shortest path between u and v has to be a chordless path. The following observations from [3] are useful tools.

Lemma 5. [3] Let $P = (w_1, w_2, \dots, w_k)$ be a chordless path in a chordal graph G and let $w_i = \text{highest}(P)$, with respect to a PEO f . Then (w_1, w_2, \dots, w_i) is an increasing path while $(w_i, w_{i+1}, \dots, w_k)$ is a decreasing path, with respect to f .

Corollary 1. Let $P = (w_1, w_2, \dots, w_k)$ be a chordless path and $w_k = \text{highest}(P)$ with respect to a PEO f . Then P is an increasing path with respect to f .

2 Internal Cuts in Chordal Graphs

Theorem 3. Let (A, B) be an internal cut of a chordal graph G with $\kappa(G) = k$. Then for each i , $0 \leq i \leq (k+1)$, there exists a clique K_i such that $|K_i| = k+1$, $|A \cap K_i| = i$ and $|B \cap K_i| = k+1-i$.

Proof: Let $x \in A$ and $y \in B$ be hidden nodes with respect to the cut (A, B) . (Since (A, B) is an internal cut, such nodes exist in A and B). Clearly x and y are nonadjacent. Then by Menger's Theorem (Theorem 1), there exist $\kappa(G) = k$ vertex disjoint paths from x to y . If there are k vertex disjoint paths, then there are k vertex disjoint chordless paths also. Let these vertex disjoint chordless paths be P'_1, P'_2, \dots, P'_k .

Clearly, $N(A)$ separates x from y . Thus there exists a minimal separator $M \subseteq N(A) \subset B$ which separates x from y . By Lemma 4, M is a clique. Then by Lemma 3, there exists a PEO such that the highest numbers are assigned to the nodes of M . Let f be such a PEO. Note that each path P'_i , $1 \leq i \leq k$, should pass through M , since M is a separator of x and y . Let z_i be the first node at which P'_i meets M . Let P_i be the partial path from x to z_i of P'_i . Clearly, $z_i = \text{highest}(P_i)$ (with respect to the PEO f), since z_i is the only node in P_i , that belongs to M , and the PEO f was selected such that the nodes in M got the highest numbers. Thus by Corollary 1, P_i is an increasing path.

Now we define a series of $k+1$ element subsets of $V(G)$, namely F_0, F_1, \dots, F_r as follows. Let $F_0 = \{x, y_1, y_2, \dots, y_k\}$ where y_i is the first node in P_i after x . Note that each $y_i \in A$, since x is a hidden node in A . Clearly each y_i has a higher PEO number than that of x , since P_i is an increasing path. Thus $\{y_1, y_2, \dots, y_k\} \subseteq N_h(x)$. Thus $F_0 = \{x, y_1, y_2, \dots, y_k\}$ induces a clique of size $k+1$, which is completely in A .

Now we will describe how to define F_{i+1} from F_i , for $0 \leq i < r$. Let $v_i = \text{lowest}(F_i)$. Define $F'_i = F_i - \{v_i\}$. Suppose that the subset F_i satisfies the following properties.

1. $|F_i| = k+1$.
2. The nodes of F_i induces a clique.
3. $F_i \subset \bigcup_{j=1}^{j=k} P_j$. Moreover, $|F'_i \cap P_j| = 1$ for each path P_j , $1 \leq j \leq k$. (That is, in F'_i , there is exactly one "representative" node from each path P_j).

Now let $u_i = \text{lowest}(F'_i) = \text{second-lowest}(F_i)$. Suppose that u_i is on path P_j . Now we define F_{i+1} from F_i as follows.

If $u_i = z_j$, the last vertex on path P_j , then let $F_i = F_r$. That is, F_i will be the last subset in the series, in that case. Otherwise, consider the node w_i , which is

placed just after u_i on the path P_j . Since P_j is an increasing path (with respect to f), $f(w_i) > f(u_i)$. Also note that $w_i \notin F_i$. To see this, note that $w_i \notin F'_i$, since $|F'_i \cap P_j| = 1$, and u_i is already there in $(F'_i \cap P_j)$. Also, $w_i \neq v_i = \text{lowest}(F_i)$, since $f(w_i) > f(u_i) > f(v_i)$. Thus w_i is a node on P_j which is not already in F_i . We define F_{i+1} as follows.

$$F_{i+1} = F_i - \{v_i\} \bigcup \{w_i\}$$

Note that F_{i+1} is obtained by replacing the node v_i , the lowest numbered node in F_i with a new node w_i . Thus $|F_{i+1}| = k + 1$. Clearly $\text{lowest}(F_{i+1}) = u_i$. Also it is clear that since F_i was assumed to induce a clique and w_i is a (higher) neighbour of u_i , $(F_{i+1} - \{u_i\}) \subseteq N_h(u_i)$, with respect to f . Thus F_{i+1} induces a clique also. Clearly $F_{i+1} \subset \bigcup_{j=1}^{j=k} P_j$. Note that $F'_{i+1} = F_{i+1} - \{u_i\} = F'_i - \{u_i\} \bigcup \{w_i\}$. In other words, F'_{i+1} is obtained by replacing u_i in F'_i by w_i . Remember that $|F'_i \cap P_j| = 1$, for each j . Since we are just replacing the “representative” node u_i of P_j (in F'_i) by another node w_i (which is also on the same path P_j), to obtain the new set F'_{i+1} , it follows that $|F'_{i+1} \cap P_j| = 1$, for each P_j , $1 \leq j \leq k$. Thus the three conditions satisfied by F_i are satisfied by F_{i+1} also.

From the inductive argument above, together with the fact that F_0 satisfies all the 3 properties stated above, it follows that each subset F_i in the sequence defined above satisfies those 3 properties.

Now we look at the last subset F_r in this sequence. Let $u_r = \text{lowest}(F'_r)$. By definition of the sequence, $u_r = z_j$, (ie, the last vertex of P_j), for some path P_j . We claim that $F'_r = \{z_1, z_2, \dots, z_k\}$. Suppose not. Then, since F'_r should contain exactly k nodes, and $F'_r \subset \bigcup_{j=1}^{j=k} P_j$, there should be a node $x_r \in F'_r$, which is on some path P_l , but not an end vertex of P_l , ie $x_r \neq z_l$. Since we had selected z_l to be the first node at which P_l meets M , $x_l \notin M$. But the PEO f was selected such that the nodes of M got the highest numbers. It follows that $f(x_l) < f(z_j)$, contradicting the assumption that $z_j = \text{lowest}(F'_r)$. Thus we infer that $F'_r = \{z_1, z_2, \dots, z_k\}$.

Clearly $|F_0 \cap A| = k + 1$, and $|F_r \cap B| \geq k$, since $F'_r = \{z_1, z_2, \dots, z_k\} \subseteq M \subseteq N(A) \subset B$. Noting that at each step, we were replacing one node in F_j with a new one to get F_{j+1} , it follows that for each value i , $1 \leq i \leq k + 1$, there should be a F_{j_i} such that $|F_{j_i} \cap A| = i$, and $|F_{j_i} \cap B| = k + 1 - i$. Let K_i be the clique induced by F_{j_i} . Clearly K_i has the desired property. Now, to see that there exists a clique K_0 such that $|K_0| = k + 1$, $|K_0 \cap A| = 0$ and $|K_0 \cap B| = k + 1$, just interchange the role of x and y and observe that just as there is a clique induced by F_0 , comprising of x and its (higher) neighbours $\{y_1, y_2, \dots, y_k\}$, which is completely in A , there exists a clique which is completely in B comprising of y and its higher neighbours. Hence the proof. ■

As our next result we show that the number of edges in an internal cut of a chordal graph is at least $\frac{\kappa(G)(\kappa(G)+1)}{2}$.

Theorem 4. *Let (A, B) be an internal cut of a chordal graph G with $\kappa(G) = k$. Then $|(A, B)| \geq \frac{k(k+1)}{2}$.*

Proof: For $\kappa(G) = 1$, the lemma is trivially true because the graph is connected and any cut will have at least one edge in it. So we consider the case when $\kappa(G) \geq 2$. Let (A, B) be an internal cut of G . By the definition of internal cuts, there exist hidden vertices $x \in A$ and $y \in B$. Since G is a k -connected graph, by Menger's theorem (Theorem 1) there are k vertex disjoint paths between x and y . Let P'_1, \dots, P'_k denote these paths. For each $P'_i, 1 \leq i \leq k$, there is a chordless path between x and y whose vertices are a subset of the vertices of P'_i . As x and y are hidden vertices, it follows that each chordless path has at least 4 vertices in it. Let these chordless paths be P_1, \dots, P_k . Clearly, these are vertex disjoint paths.

Let $(u_1, v_1), \dots, (u_k, v_k)$ be the edges from P_1, \dots, P_k , respectively, such that for $1 \leq i \leq k$, $u_i \in A$ and $v_i \in B$. An edge is said to cross a cut if its end points are not on the same side of the cut. The theorem is proved by showing that for every pair of paths $\{P_i, P_j\}, 1 \leq i \neq j \leq k$, there is an edge e_{ij} crossing the cut (A, B) , one of whose end points is on $P_i - \{x, y\}$, and the other on $P_j - \{x, y\}$. As P_1, \dots, P_k are vertex disjoint paths between x and y , an edge cannot be associated with two distinct pairs of paths. This counts $\binom{k}{2}$ edges crossing the cut, which when considered along with the edges $\{u_i, v_i\}, 1 \leq i \leq k$, yields $| (A, B) | \geq \frac{k(k+1)}{2} = \frac{\kappa(G)(\kappa(G)+1)}{2}$.

Let P_i, P_j be two distinct paths from the set $\{P_1, \dots, P_k\}$. We claim that there is an edge e_{ij} crossing the cut (A, B) , one of whose end points is on $P_i - \{x, y\}$, and the other on $P_j - \{x, y\}$. Let us assume the contrary and prove our claim by arriving at a contradiction. Let w_1 be the last vertex in A on P_i which has an edge to a vertex on P_j . (Note that $w_1 \neq x$, since the node appearing just after x , on P_i , itself has a neighbour on P_j , namely x). By our assumption that no chord starting from a node on P_i and ending at a node on P_j crosses the cut, $N(w_1) \cap P_j \subseteq A$. In P_i , on the path from w_1 to y , let w_2 be the first vertex in B that has an edge to a vertex on P_j . Clearly, $w_2 \neq y$ and by our assumption, $N(w_2) \cap P_j \subseteq B$. Let w_4 and w_3 be the closest vertices on P_j such that $w_4 \in N(w_1) \cap P_j$ and $w_3 \in N(w_2) \cap P_j$. Clearly, the portion of the path P_j from w_3 to w_4 will not contain any other vertices of $N(w_1)$ or $N(w_2)$. Now, the path from w_1 to w_2 , the edge (w_2, w_3) , the path from w_3 to w_4 , and the edge (w_4, w_1) form an induced cycle of length at least 4. (This easily follows from the construction and the assumption that P_i and P_j are chordless paths). This is clearly a contradiction since G is a chordal graph. Therefore, our assumption is wrong and there exists an edge e_{ij} which crosses the cut (A, B) , with one end point on P_i and the other end point on P_j . Finally, such an edge cannot have x or y as one of its end points, since that would violate the assumption that P_i and P_j are chordless. Hence the theorem. ■

The above lower bound for the size of an internal cut of a chordal graph can be tight. Consider the following graph for instance.

Example 1: Construct a graph G as follows. Let A be a clique of N_1 nodes and B a clique of N_2 nodes. Let u_1, u_2, \dots, u_k be k nodes in A and v_1, v_2, \dots, v_k be k nodes in B . Add an edge from v_i ($1 \leq i \leq k$), to each node in the set

$\{u_i, u_{i+1}, \dots, u_k\}$. Thus, v_1 is connected to $\{u_1, u_2, \dots, u_k\}$, v_2 is connected to $\{u_2, u_3, \dots, u_k\}$ etc.

To verify that the above graph is a chordal graph, we describe a PEO f for this graph. Let the last N_1 numbers (i.e., the numbers from N_2+1 to N_2+N_1) be given to the nodes of A in some order. Now let $f(v_i) = N_2 - i + 1$, for $1 \leq i \leq k$. Let the remaining nodes in B get the numbers from 1 to $N_2 - k$ in any order. It is easy to verify that this ordering is a PEO. It follows that G is a chordal graph. (See Theorem 2).

Suppose that $N_1, N_2 > k$. Then it is easy to verify that $\kappa(G) = k$. (A, B) is an internal cut since A contains nodes which are not adjacent to any node in B and B contains nodes which are not adjacent to any node in A . Clearly $|(A, B)| = k + k - 1 + \dots + 1 = \frac{k(k+1)}{2}$. ■

3 Edge Connectivity vs. Vertex Connectivity in Chordal Graphs

In this section we show that if for a chordal graph G , $\lambda(G) < \delta(G)$, then $\lambda(G) \geq \frac{k(k+1)}{2}$, where $\kappa(G) = k$.

Lemma 6. *Let (A, B) be a mincut of a connected undirected graph G . Then $G(A)$ and $G(B)$ are connected.*

Proof: Suppose that $G(A)$ is not connected. Let $G(A_1)$ be a connected component of $G(A)$, where $A_1 \subset A$. Let $A_2 = A - A_1$. Clearly $(A_1, B \cup A_2)$ is a cut of G and $|(A_1, B \cup A_2)| < |(A, B)|$, since $(A_1, B \cup A_2) \subset (A, B)$. But this is a contradiction since (A, B) is assumed to be the mincut. ■

Lemma 7. *Let (A, B) be a mincut of G , and let $\lambda(G) = |(A, B)| < \delta(G)$. Then there exists a node $x \in A$, such that $x \notin N(B)$. Similarly $\exists y \in B$ such that $y \notin N(A)$. That is (A, B) is an internal cut.*

Proof: Suppose that every node in A is adjacent to some node in B . Let u be a node in A . Let $F = (A, B)$, be the minimum cut. We have

$$d(u) = |N(u) \cap B| + |N(u) \cap A| \leq |N(u) \cap B| + |A| - 1$$

But

$$|F| = \sum_{x \in A} |N(x) \cap B| \geq |N(u) \cap B| + |A| - 1$$

since each term in the sum should be at least 1 by the assumption that every node in A is adjacent to at least one node in B . It follows that $d(u) \leq |F|$. But by assumption, $d(u) \geq \delta(G) > |F|$. Thus we have a contradiction and we conclude that there is a node $x \in A$ such that $x \notin N(B)$. By similar arguments, $\exists y \in B$ such that $y \notin N(A)$. ■

Theorem 5. *For a chordal graph G , if $\lambda(G) \neq \delta(G)$, then $\lambda(G) \geq \frac{(\kappa(G))(\kappa(G)+1)}{2}$*

Proof: By Lemma 7, when $\lambda(G) < \delta(G)$, the mincut will be an internal cut. Then the result follows from Theorem 4 ■

The above theorem is strict. Consider the graph G in Example 1. If $N_1, N_2 > \frac{k(k+1)}{2}$, then it is clear that $\delta(G) > \frac{k(k+1)}{2} = |(A, B)| \geq \lambda(G)$. In this case it is easy to verify that (A, B) is in fact the mincut. Thus $\lambda(G) = \frac{\kappa(G)(\kappa(G)+1)}{2}$, for this graph.

References

1. P. BUNEMAN, *A characterisation of rigid circuit graphs*, Discrete Mathematics, 9 (1974), pp. 205–212.
2. L. S. CHANDRAN, *Edge connectivity vs vertex connectivity in chordal graphs*, in Proceedings of the 7th International Computing and Combinatorics conference, LNCS, 2001.
3. ———, *A linear time algorithm for enumerating all the minimum and minimal separators of a chordal graph*, in Proceedings of the 7th International Computing and Combinatorics Conference, LNCS 2108, Springer, Berlin, 2001, pp. 308–317.
4. L. S. CHANDRAN, T. KAVITHA, AND C. R. SUBRAMANIAN, *Isoperimetric inequalities and the width parameters of graphs*, in Proceedings of the 9th International Computing and Combinatorics Conference, LNCS 2697, 2003, pp. 385–395.
5. R. CHANDRASEKARAN AND A. TAMIR, *Polynomially bounded algorithms for locating p -centres on a tree*, Math. Programming, 22 (1982), pp. 304–315.
6. G. CHARTRAND AND F. HARARY, *Graphs with prescribed connectivities*, in Theory of Graphs, P. Erdős and G. Katona, eds., Akademiai Kiado, Budapest, 1968, pp. 61–63.
7. M. C. GOLUMBIC, *Algorithmic Graph Theory And Perfect Graphs*, Academic Press, New York, 1980.
8. F. HARARY, *Graph Theory*, Addison-Wesley Reading, MA, 1969.
9. T. KLOKS, *Treewidth: Computations And Approximations*, vol. 842 of Lecture Notes In Computer Science, Springer Verlag, Berlin, 1994.
10. R. H. MOHRING, *Computational Graph Theory*, Springer, Wein, New York, 1990, ch. Graph Problems Related To Gate Matrix Layout And PLA Folding, pp. 17–52.
11. C. PAPADIMITRIOU AND M. YANNAKAKIS, *Scheduling interval ordered tasks*, SIAM Journal of Computing, 8 (1979), pp. 405–409.
12. D. ROSE, *Triangulated graphs and the elimination process*, J. Math. Ana. Appl., 32 (1970), pp. 597–609.
13. ———, *Graph Theory and Computing*, Academic Press, New York, 1972, ch. A graph theoretic study of the numerical solution of sparse positive definite systems of linear equations, pp. 183–217.
14. H. WHITNEY, *Congruent graphs and the connectivity of graphs*, American J. Math., 54 (1932), pp. 150–168.
15. M. YANNAKAKIS, *Computing the minimum Fill-in is NP-complete*, SIAM J. on Alge. Discre. Math., 2 (1981), pp. 77–79.

The Worst-Case Time Complexity for Generating All Maximal Cliques*

(Extended Abstract)

Etsuji Tomita, Akira Tanaka**, and Haruhisa Takahashi

Department of Information and Communication Engineering,
The University of Electro-Communications,
Chofugaoka 1-5-1, Chofu, Tokyo 182-8585, Japan
tomita@ice.uec.ac.jp

Abstract. We present a depth-first search algorithm for generating all maximal cliques of an undirected graph, in which pruning methods are employed as in Bron and Kerbosch's algorithm. All maximal cliques generated are output in a tree-like form. Then we prove that its worst-case time complexity is $O(3^{n/3})$ for an n -vertex graph. This is optimal as a function of n , since there exist up to $3^{n/3}$ cliques in an n -vertex graph.

1 Introduction

In an undirected graph G , a *clique* is a complete subgraph of G in which any two vertices are adjacent. The set of vertices of a maximal clique of the complementary graph of G is a *maximal independent set* of G . Generating maximal cliques or maximal independent sets of a given graph is one of the fundamental problems in the theory of graphs and has many diverse applications, e.g., for clustering and for bioinformatics [10], [7]. A number of algorithms have been presented and evaluated experimentally or theoretically for this problem; see, e.g., [5], [14], [9], [6] and [4], where [4] is an elaborate survey paper on the maximum clique problem. Among them, Bron and Kerbosch [5] presented an algorithm for generating *all* maximal cliques of a graph together with some experimental results. Tsukiyama *et al.* [14] devised an algorithm for generating all maximal independent sets in a graph G in $O(nm\mu)$ -time, where n , m , and μ are the numbers of vertices, edges, and maximal independent sets of G , respectively. Lawler *et al.* [9] generalized this result further. Chiba and Nishizeki [6] improved Tsukiyama *et al.*'s algorithm and gave a more efficient algorithm for listing all maximal cliques of G in $O(a(G)m\mu)$ -time, where $a(G)$ is the arboricity of G with $a(G) \leq O(m^{1/2})$ for a connected graph G , and μ is the number of maximal cliques in G .

* This research has been supported in part by Grants-in-Aid for Scientific Research Nos. 13680435 and 16300001 from the Ministry of Education, Culture, Sports, Science and Technology, Japan, and Research Fund of the University of Electro-Communications. It is also given a grant by Funai Foundation for Information Technology.

** Presently with Toyota Techno Service Corporation, Imae 1-21, Hanamotouchou, Toyota, Aichi 470-0334, Japan.

We present here a depth-first search algorithm for generating all maximal cliques of an undirected graph, in which pruning methods are employed as in Bron and Kerbosch's algorithm [5]. All the cliques generated are output in a tree-like form. Then we prove that its worst-case running time complexity is $O(3^{n/3})$ for a graph with n vertices. This is the best one could hope for as a function of n , since there exist up to $3^{n/3}$ maximal cliques in a graph with n vertices as shown by Moon and Moser [11]. An earlier version of this paper appeared in [13].

2 Preliminaries

[1] Throughout this paper, we are concerned with a simple undirected *graph* $G = (V, E)$ with a finite set V of *vertices* and a finite set E of *unordered* pairs (v, w) of distinct vertices, called *edges*. A pair of vertices v and w are said to be *adjacent* if $(v, w) \in E$. We call $\overline{G} = (V, \overline{E})$ with $\overline{E} = \{(v, w) \in V \times V \mid v \neq w, \text{ and } (v, w) \notin E\}$ the *complementary* graph of G .

[2] For a vertex $v \in V$, let $\Gamma(v)$ be the set of all vertices which are adjacent to v in $G = (V, E)$, i.e., $\Gamma(v) = \{w \in V \mid (v, w) \in E\}$ ($\nexists v$).

[3] For a subset $W \subseteq V$ of vertices, $G(W) = (W, E(W))$ with $E(W) = \{(v, w) \in W \times W \mid (v, w) \in E\}$ is called a *subgraph* of $G = (V, E)$ *induced* by W . For a set W of vertices, $|W|$ denotes the number of elements in W .

[4] Given a subset $Q \subseteq V$ of vertices, the induced subgraph $G(Q)$ is said to be *complete* if $(v, w) \in E$ for all $v, w \in Q$ with $v \neq w$. If this is the case, we may simply say that Q is a complete subgraph. A complete subgraph is also called a *clique*. If a clique is not a proper subgraph of another clique then it is called a *maximal* clique. A subset $W \subseteq V$ of vertices is said to be *independent* if $(v, w) \notin E$ for all $v, w \in W$. Here, $Q \subseteq V$ is a maximal clique of G if and only if Q is a maximal independent set of the complementary graph \overline{G} .

3 The Algorithm

We consider a depth-first search algorithm for generating all the cliques of a given graph $G = (V, E)$ ($V \neq \phi$).

Here we introduce a global variable Q of a set of vertices which constitute a complete subgraph being found up to this time. Then we begin the algorithm by letting $Q := \phi$, and expand it step by step by applying a recursive procedure EXPAND to V and its succeeding induced subgraphs searching for larger and larger complete subgraphs until they reach maximal ones.

Let $Q = \{p_1, p_2, \dots, p_d\}$ be found to be a complete subgraph at some stage, and consider a *subgraph* $G(\text{SUBG})$ which is induced by a set of vertices

$$\text{SUBG} = V \cap \Gamma(p_1) \cap \Gamma(p_2) \cap \dots \cap \Gamma(p_d),$$

where $\text{SUBG} = V$ and $Q = \phi$ at the initial stage. Then apply the procedure EXPAND to SUBG searching for larger complete subgraphs. If $\text{SUBG} = \phi$,

then Q is clearly a maximal complete subgraph, or, a maximal clique. Otherwise, $Q \cup \{q\}$ is a larger complete subgraph for every $q \in SUBG$. Then consider smaller subgraphs $G(SUBG_q)$ which are induced by new sets of vertices

$$SUBG_q = SUBG \cap \Gamma(q)$$

for all $q \in SUBG$, and apply recursively the same procedure EXPAND to $SUBG_q$ to find larger complete subgraphs containing $Q \cup \{q\}$.

Thus far we have shown only the basic framework of the algorithm for generating all maximal cliques (with possible duplications). This process can be represented by the following search forest, or the collection of search trees: The set of roots of the search forest is exactly the same as V of graph $G = (V, E)$. For each $q \in SUBG$, all vertices in $SUBG_q = SUBG \cap \Gamma(q)$ are sons of q . Thus, a set of vertices along a path from the root to any vertex of the search forest constitutes a complete subgraph, or a clique. We shall show an example of a search forest (with unnecessary subtrees deleted) later in Fig. 2 (b).

Now we proceed to describe two methods to prune unnecessary parts of the search forest. We regard the previously described set $SUBG (\neq \phi)$ to be an *ordered* set of vertices, and we continue to generate maximal cliques from vertices in $SUBG$ step by step in this order .

First, let $FINI$ be the *first half* of $SUBG$ and suppose that we have already *finished* expanding search subtrees from every vertex $q' \in FINI \subseteq SUBG$ to generate *all* maximal cliques containig $Q \cup \{q'\}$, and that only the remaining vertex $q \in CAND = SUBG - FINI$ is a *candidate* for further expansion of the present complete subgraph Q to generate new larger cliques. That is, let

$$SUBG = FINI \cup CAND \quad (FINI \cap CAND = \phi),$$

with

$$FINI = \{q' \in SUBG \mid \text{The procedure has finished generating} \\ \text{all maximal cliques containing } Q \cup \{q'\}\}.$$

Consider the subgraph $G(SUBG_q)$ with $SUBG_q = SUBG \cap \Gamma(q)$, and let

$$SUBG_q = FINI_q \cup CAND_q \quad (FINI_q \cap CAND_q = \phi),$$

where

$$FINI_q = FINI \cap \Gamma(q), \text{ and } CAND_q = CAND \cap \Gamma(q).$$

Then only the vertices in the subgraph $G(CAND_q)$ can be candidates for expanding the complete subgraph $Q \cup \{q\}$ to find *new* larger cliques, since all the cliques containing $(Q \cup \{q\}) \cup \{r\}$ with $r \in FINI_q \subseteq FINI$ have already been generated for any r by application of the procedure EXPAND to $FINI$ as stated above. Thus, further expansion is to be considered only for vertices in $G(CAND_q)$ excluding ones in $FINI_q = SUBG_q - CAND_q$.

Secondly, given a certain vertex $u \in SUBG$, suppose that *all* the maximal cliques containing $Q \cup \{u\}$ have been generated. Then every *new* maximal clique

containing Q , but not $Q \cup \{u\}$, should contain at least one vertex $v \in SUBG - \Gamma(u)$. This is because if Q is expanded to a complete subgraph $R \subseteq (Q \cup S) \cap (SUBG - \{u\})$ with $S \subseteq \Gamma(u)$, then $R \cup \{u\}$ is a *larger* complete subgraph, and hence R is not *maximal*. Thus, any new maximal clique can be found by expanding Q to $Q \cup \{q\}$ such that $q \in SUBG - \Gamma(u)$, and by generating all the cliques containing $Q \cup \{q\}$. Therefore, if we have expanded a search subtree from $u \in SUBG$, then we should not expand any search subtree from $w \in SUBG \cap \Gamma(u)$.

Taking the previous pruning method also into consideration, the only search subtrees to be expanded are from vertices in $(SUBG - SUBG \cap \Gamma(u)) - FINI = CAND - \Gamma(u) (\ni u)$. Here, in order to minimize $|CAND - \Gamma(u)|$, we choose such vertex $u \in SUBG$ to be the one which maximizes $|CAND \cap \Gamma(u)|$. In this way, the problem of generating all maximal cliques of $G(CAND)$ can be decomposed into $k = |CAND - \Gamma(u)|$ such subproblems; see **Lemma 1** (i) in Section 4.

Now we present an algorithm CLIQUES for generating all maximal cliques without duplications in Fig. 1.

If and only if Q is found to be a *maximal* clique at statement 2, we only print out a string of characters “*clique*,” instead of Q itself at statement 3. Otherwise, it is impossible to achieve the worst-case running time of $O(3^{n/3})$ for an n -vertex graph, since printing out Q itself requires time proportional to the size of Q . Instead, in addition to statement 3, not only do we print out q followed by a *comma* at statement 7 every time q is picked out as a new element of a larger clique, but we also print out a string of characters “*back*,” at statement 12 after q is moved from $CAND$ to $FINI$ at statement 11. We can easily obtain a tree representation of all the maximal cliques from the resulting sequence printed by statements 3, 7, and 12. Here, primed statements 0', 7', and 12' are only for the sake of explanation.

Example. Let us apply the above algorithm CLIQUES to a graph in Fig. 2 (a). Then the whole process is represented by a search forest in Fig. 2 (b), and we show the resulting printed sequence in Fig. 2 (c) *with appropriate indentations*. In Fig. 2 (b), each set of vertices surrounded by a flat circle represents $SUBG$ at that stage, in which vertex with \triangle mark is in $FINI \subseteq SUBG$ at the beginning. Vertex u chosen at statement 4 is marked by \square or \boxtimes depending on whether it is in $CAND$ or $FINI$, respectively. Other vertices in $CAND - \Gamma(u)$ are marked by \circ , while vertices in $CAND \cap \Gamma(u)$ are marked by \bullet . In conclusion, all the maximal cliques of G are $\{4, 6, 7, 8\}$, $\{4, 6, 5\}$, $\{4, 3, 8\}$, $\{1, 2, 9\}$, and $\{2, 3, 9\}$. \square

Now given only the resulting printed sequence in Fig. 2 (c) without indentations, we can easily obtain essentially the same result as above by reconstructing from it a tree which represents a principal part of the previous search forest in Fig. 2 (b). Here, a dot “.” ($\notin V$) is introduced as a virtual root of the tree at the beginning. Then every time “ q ,” is encountered in the sequence, we expand a downward edge whose end point is labeled by q . If and only if “ q ,” is followed by “*clique*,” the set of all the vertices along the path from the root to the vertex q excluding the root (\cdot) represents a maximal clique. Every time “*back*,” is

```

procedure CLIQUES( $G$ )
    /* Graph  $G = (V, E)$  */
begin
    0':/*  $Q := \phi$  */
    /* global variable  $Q$  is to constitute a clique */
    1: EXPAND( $V, V$ )
        procedure EXPAND( $SUBG, CAND$ )
            begin
            2:   if  $SUBG = \phi$ 
            3:   then print ("clique,")
                /* to represent that  $Q$  is a maximal clique */
            4:   else  $u :=$  a vertex  $u$  in  $SUBG$  which maximizes  $|CAND \cap \Gamma(u)|$ 
                /* let  $EXT_u = CAND - \Gamma(u)$  */
            5:   while  $CAND - \Gamma(u) \neq \phi$ 
            6:   do  $q :=$  a vertex in  $(CAND - \Gamma(u))$ 
            7:   print ( $q, ", "$ )
                /* to represent the next statement */
            7':/*  $Q := Q \cup \{q\};$  */
            8:    $SUBG_q := SUBG \cap \Gamma(q);$ 
            9:    $CAND_q := CAND \cap \Gamma(q);$ 
            10:  EXPAND( $SUBG_q, CAND_q$ );
            11:   $CAND := CAND - \{q\}$  /*  $FINI := FINI \cup \{q\}$  */
            12:  print ("back,")
                /* to represent the next statement */
            12':/*  $Q := Q - \{q\}$  */
            od
            fi
        end of EXPAND
    end of CLIQUES

```

Fig. 1. Algorithm CLIQUES

encountered in the sequence, we go up the tree backward by one edge to find other maximal cliques. It is clear that this transformation can be done in time proportional to the length of the resulting sequence.

4 The Worst-Case Time Complexity

Given $G = (V, E)$ with $V \neq \phi$, we evaluate the worst-case running time of the previous algorithm CLIQUES(G) with the primed statements 0', 7', and 12' having been deleted. So, this is equivalent to evaluating that of EXPAND(V, V).

Now we begin by giving a few definitions.

- [1] Let $T(n, m)$ be an upper bound on the worst-case running time of EXPAND($SUBG, CAND$) when $|SUBG| = n$ and $|CAND| = m$ ($n \geq m \geq 1$).
- [2] Let $T_k(n, m)$ be an upper bound on the worst-case running time of EXPAND($SUBG, CAND$) when $|SUBG| = n$, $|CAND| = m$, and $|EXT_u| = |CAND - \Gamma(u)| = k$ at the first entrance to statement 5.

From the above definitions, we have that

$$T(n, m) = \max_{1 \leq k \leq m} \{T_k(n, m)\}. \quad (1)$$

The following Lemma is a key for evaluating $T(n, m)$.

Lemma 1. Consider $\text{EXPAND}(\text{SUBG}, \text{CAND})$ when $|\text{SUBG}| = n$, $|\text{CAND}| = m$, $|\text{EXT}_u| = |\text{CAND} - \Gamma(u)| = k \neq 0$, and $|\text{CAND} \cap \Gamma(u)| = m - k$ at the first entrance to statement 5. In what follows, CAND stands exclusively for this initial value, though it is decreased one by one at statement 11 in the **while** loop. Let $\text{CAND} - \Gamma(u) = \{v_1, v_2, \dots, v_k\}$ and the vertex at statement 6 be chosen in this order. Let

$$\text{SUBG}_{v_i} = \text{SUBG} \cap \Gamma(v_i),$$

$$\text{CAND}_{v_i} = \text{CAND} \cap \Gamma(v_i), \text{ and}$$

$$\text{CAND}_i = (\text{CAND} - \{v_1, v_2, \dots, v_{i-1}\}) \cap \Gamma(v_i).$$

Then the following hold.

- (i) $T_k(|\text{SUBG}|, |\text{CAND}|) \leq \sum_{i=1}^k T(|\text{SUBG}_{v_i}|, |\text{CAND}_i|) + P(n)$.
- (ii) a) $|\text{CAND}_i| \leq |\text{CAND}_{v_i}| \leq m - k$.
b) $|\text{SUBG}_{v_i}| \leq n - k \leq n - 1$.

Proof. (i) It is obvious from **procedure** $\text{EXPAND}(\text{SUBG}, \text{CAND})$ and the definition of $P(n)$.

- (ii) a) $|\text{CAND}_i| \leq |\text{CAND}_{v_i}| = |\text{CAND} \cap \Gamma(v_i)| \leq |\text{CAND} \cap \Gamma(u)| = m - k$.
b) $|\text{SUBG}_{v_i}| = |\text{SUBG} \cap \Gamma(v_i)|$
 $= |\text{FINI} \cap \Gamma(v_i)| + |\text{CAND} \cap \Gamma(v_i)|$,
 with $|\text{FINI} \cap \Gamma(v_i)| \leq |\text{FINI}| = n - m$ and $|\text{CAND} \cap \Gamma(v_i)| \leq m - k$.
 Here, $(n - m) + (m - k) = n - k \leq n - 1$, since $k \geq 1$. **Q.E.D.**

Theorem 1. For the upper bound $T(n, m)$ of the worst-case running time of $\text{EXPAND}(\text{SUBG}, \text{CAND})$ with $|\text{SUBG}| = n$ and $|\text{CAND}| = m$, the following holds for all $n \geq m \geq 1$:

$$T(n, m) \leq C3^{n/3} - Q(n) \equiv R(n), \quad (2)$$

where

$$Q(n) = q_1 n^2 + q_2 n + q_3,$$

with

$$q_1 = p_1/2 > 0, \quad q_2 = 9p_1/2 > 0, \quad q_3 = 27p_1/2 > 0,$$

and

$$C = \text{Max}\{C_1, C_2, C_3\},$$

with $C_1 = 3q_2/\ln 3 = 27p_1/2 \ln 3$, $C_2 = 39p_1/(2 \cdot 3^{1/3})$, and C_3 being the maximum over n of $3(1 - 2 \cdot 3^{-2/3})^{-1} \cdot Q(n - 3)/3^{n/3}$. (Note that $Q(n - 3)/3^{n/3}$ is bounded above, since it approaches 0 as n tends to infinity. Hence, C_3 is well-defined.)

Here, $R(n) \equiv C3^{n/3} - Q(n)$ is monotone increasing with $R(n) \geq 0$ for all integers $n \geq 0$.

Proof. First, by differentiating a continuous function $R(x)$ with x being a real number and $C \geq C_1$, we can prove that $R(n)$ is monotone increasing for all integers $n \geq 0$. Furthermore, $R(1) = C3^{1/3} - Q(1) \geq C_23^{1/3} - 37p_1/2 = p_1$, since $C \geq C_2 = 39p_1/(2 \cdot 3^{1/3})$. So, $R(n) \geq p_1$ for all integers $n \geq 1$.

Now we prove that Eq.(2) holds by induction on n .

Basis. $n(=m) = 1$. We have $T(1, 1) = P(1) = p_1$ by the definition of $P(n)$. So, Eq.(2) holds for $n = m = 1$, since $R(1) \geq p_1$.

Induction step. We assume that Eq.(2) holds for all integers n, m , $1 \leq m \leq n \leq N$, and prove that it also holds for $1 \leq m \leq n = N + 1$.

Consider $\text{EXPAND}(\text{SUBG}, \text{CAND})$ when $|\text{SUBG}| = n = N + 1$, $|\text{CAND}| = m$ ($1 \leq m \leq n = N + 1$), $|\text{EXT}_u| = |\text{CAND} - \Gamma(u)| = k \neq 0$ with $\text{CAND} - \Gamma(u) = \{v_1, v_2, \dots, v_k\}$ at the first entrance to statement 5. Then just as in **Lemma 1** (i), we have

$$\begin{aligned} T_k(n, m) &= T_k(|\text{SUBG}|, |\text{CAND}|) \\ &\leq \sum_{i=1}^k T(|\text{SUBG}_{v_i}|, |\text{CAND}_i|) + P(n), \end{aligned}$$

where $|\text{SUBG}_{v_i}| \leq n - 1 = N$ by **Lemma 1** (ii)-b). Then by the induction hypothesis we have

$$\sum_{i=1}^k T(|\text{SUBG}_{v_i}|, |\text{CAND}_i|) \leq \sum_{i=1}^k R(|\text{SUBG}_{v_i}|).$$

Since $R(n)$ is monotone increasing and $|\text{SUBG}_{v_i}| \leq n - k$, we have

$$\sum_{i=1}^k R(|\text{SUBG}_{v_i}|) \leq kR(n - k).$$

Combining these inequalities followed by some calculations yields

$$\begin{aligned} T_k(n, m) &\leq C3^{n/3} - \{3Q(n - 3) - P(n)\} \\ &= C3^{n/3} - Q(n) \quad (\text{from the definition of } Q(n)). \end{aligned}$$

Substituting this inequality into Eq.(1), we have

$$T(n, m) \leq C3^{n/3} - Q(n).$$

Thus, Eq.(2) also holds for $n = N + 1 \geq m \geq 1$. Therefore, Eq.(2) has been induced to hold for *all* integers $n \geq m \geq 1$. Hence the result. **Q.E.D.**

In particular,

$$T(n, n) \leq C3^{n/3} - Q(n).$$

Therefore, we conclude that the worst-case running time of the algorithm $\text{CLIQUES}(G)$ is $O(3^{n/3})$ for an n -vertex graph $G = (V, E)$.

Note here that if we output a list of all the individual maximal cliques, it takes $O(n3^{n/3})$ time in the worst case.

5 Concluding Remarks

We have given an algorithm for generating all maximal cliques in a graph of n -vertices and have proved that its worst-case time complexity is $O(3^{n/3})$ that is optimal as a function of n . It is regarded to be very hard to analyze theoretically the time complexity of our algorithm as a function of the number of maximal cliques in the graph.

The algorithm CLIQUES is very simple and is easy to be implemented. It is demonstrated by our computational experiments that CLIQUES runs much faster than the sophisticated algorithm CLIQUE [6] for several random graphs and Moon Moser graphs. In particular, CLIQUES is very much faster than CLIQUE for non-sparse graphs in which the number of maximal cliques is very large. In our experiments, for example, our algorithm CLIQUES runs more than 100 times faster than CLIQUE for random graphs of 180 vertices with edge density = 0.5, and for a Moon Moser graph of 45 vertices.

Our present depth-first search algorithm together with some heuristics yields an efficient algorithm for finding *a maximum* clique [12] that is experimentally demonstrated to outperform in general other existing such algorithms. It has been successfully applied to interesting problems in bioinformatics [1]–[3], the design of DNA and RNA sequences for bio-molecular computation [8], and others.

Note here that the algorithm in [12] for finding *a maximum* clique does not select a vertex with the maximum degree in each recursive call, which is in contrast to statement 4 in Fig. 1 of the present algorithm CLIQUES for generating *all* maximal cliques. This is because the algorithm [12] with a simple bounding rule by heuristics for finding *only one* maximum clique in each recursive call takes less time than algorithms which are based on such a strategy as above for CLIQUES. In our present problem, however, we can not use such a bounding rule for finding *only one* maximum clique. In addition, *theoretical* time-complexity of an algorithm with heuristics is very hard to analyze. This is the reason why we employ here the strategies as in Bron and Kerbosch's algorithm.

Acknowledgment

The authors would like to acknowledge useful comments by J. Tarui, T. Akutsu, E. Harley, and the referees.

References

1. T. Akutsu, M. Hayashida, E. Tomita, J. Suzuki, and K. Horimoto: Protein threading with profiles and constraints, Proc. IEEE Symp. on Bioinformatics and Bio-engineering (2004, to appear).
2. D. Bahadur K. C., T. Akutsu, E. Tomita, T. Seki, and A. Fujiyama: Point matching under non-uniform distortions and protein side chain packing based on efficient maximum clique algorithms, Genome Informatics 13 (2002) 143–152.

3. D. Bahadur K. C., T. Akutsu, E. Tomita, and T. Seki: Protein side-chain packing: A maximum edge weight clique algorithmic approach, *Proc. Asia-Pacific Bioinformatics Conf.* (2004) 191–200.
4. I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo: The maximum clique problem. In: D.-Z. Du and P. M. Pardalos (eds.). *Handbook of Combinatorial Optimization*, Supplement vol. A, Kluwer Academic Publishers (1999) 1–74 .
5. C. Bron and J. Kerbosch: Algorithm 457, Finding all cliques of an undirected graph, *Comm. ACM* 16 (1973) 575–577.
6. N. Chiba and T. Nishizeki: Arboricity and subgraph listing algorithms, *SIAM J. Comput.* 14 (1985) 210–223.
7. M. Hattori, Y. Okuno, S. Goto, and M. Kanehisa: Development of a chemical structure comparison method for integrated analysis of chemical and genomic information in the metabolic pathways. *J. Am. Chem. Soc.* 125 (2003) 11853–11865.
8. S. Kobayashi, T. Kondo, K. Okuda, and E. Tomita: Extracting globally structure free sequences by local structure freeness, *Proc. DNA9* (2003) 206.
9. E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan: Generating all maximal independent sets, NP-hardness and polynomial-time algorithms, *SIAM J. Comput.* 9 (1980) 558–565.
10. S. Mohseni-Zadeh, A. Louis, P. Brézellec, and J.-L. Risler: PHYTOPROT: a database of clusters of plant proteins, *Nucleic Acids Res.* 32 (2004) D351–D353.
11. J. W. Moon and L. Moser: On cliques in graphs, *Israel J. Math.* 3 (1965) 23–28.
12. E. Tomita and T. Seki: An efficient branch-and-bound algorithm for finding a maximum clique, *DMTCS 2003, Lec. Notes in Comput. Sci.* 2731 (2003) 278–289.
13. E. Tomita, A. Tanaka, and H. Takahashi: An optimal algorithm for finding all the cliques, *Technical Report of IPSJ*, 1989-AL-12 (1989) 91–98.
14. S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa: A new algorithm for generating all the maximal independent sets, *SIAM J. Comput.* 6 (1977) 505–517.

Regular Expressions for Languages over Infinite Alphabets (Extended Abstract)

Michael Kaminski¹ and Tony Tan²

¹ Department. of Computer Science, Technion – Israel Institute of Technology,
Haifa 32000, Israel

`kaminski@cs.technion.ac.il.`

² School of Computing, National University of Singapore,

3 Science Drive 2, Singapore 117543

`tantony@comp.nus.edu.sg`

Abstract. In this paper we introduce a notion of a *regular expression* over *infinite alphabets* and show that a language is definable by an infinite alphabet regular expression if and only if it is acceptable by *finite-state unification based automaton* – a model of computation that is tightly related to other models of automata over infinite alphabets.

1 Introduction

A new model of a finite-state automata dealing with *infinite alphabets*, called *finite-state datalog automata* (FSDA) was introduced in [7]. FSDA were intended for the abstract study of relational languages. Since the character of relational languages requires the use of infinite alphabets of names of variables, in addition to a finite set of states, FSDA are equipped with a finite set of “registers” capable of retaining a variable name (out of an infinite set of names). The equality test, which is performed in ordinary finite-state automata (FA) was replaced with *unification*, which is a crucial element of relational languages.

Later, FSDA were extended in [3] to a more general model dealing with infinite alphabets, called *finite-memory automata* (FMA). FMA were designed to accept the counterpart of regular languages over infinite alphabets. Similarly to FSDA, FMA are equipped with a finite set of registers which are either empty or contain a symbol from the infinite alphabet, but contrary to FSDA, registers in FMA cannot contain values currently stored in other registers. By restricting the power of the automaton to copying a symbol to a register and comparing the content of a register with an input symbol only, without the ability to perform *any* functions, the automaton is only able to “remember” a finite set of input symbols. Thus, the languages accepted by FMA possess many of the properties of regular languages.

Whereas decision of the emptiness and containment for FMA- (and, consequently, for FSDA-) languages is relatively simple, the problem of inclusion for FMA-languages is undecidable, see [6].

An extension of FSDA to a general infinite alphabet called *finite-state unification based automata*, (FSUBA) was proposed in [8]. These automata are similar

in many ways to FMA, but are a bit weaker, because a register of FSUBA may contain values currently stored in other registers. It was shown in [8] that FSDA can be simulated by FSUBA and that the problem of inclusion for FSUBA languages is decidable.

While the study of finite automata over infinite alphabets started as purely theoretical, since the appearance of [3] and [4] it seems to have turned to more practically oriented. The key idea for the applicability is finding practical interpretations to the infinite alphabet and to the languages over it.

- In [6], members of (the infinite) alphabet Σ are interpreted as records of *communication actions*, “send” and “receive” of messages during inter-process-communication, words in a language L over this alphabet are MSCs, message sequence charts, capturing behaviors of the communication network.
- In [1], members of Σ are interpreted as URLs’ addresses of internet sites, a word in L is interpreted as a “navigation path” in the internet, the result of some finite sequence of clicks.
- In [2] there is another internet-oriented interpretation of Σ , namely, XML mark-ups of pages in a site.

In this paper we introduce a notion of a *regular expression* for languages over infinite alphabets and show that a language is definable by an infinite alphabet regular expression if and only if it is acceptable by an FSUBA.

The paper is organized as follows. In the next section we recall the definition of FSUBA and in Section 3 we present the main result of our paper – *unification based regular expressions* for languages over infinite alphabets, whose equivalence to FSUBA is proven in the Section 4.

2 Finite-State Unification Based Automata

Till the end of this paper Σ is an infinite alphabet not containing $\#$ that is reserved to denote an empty register. For a word $\mathbf{w} = w_1w_2 \cdots w_r$ over $\Sigma \cup \{\#\}$, we define the *content* of \mathbf{w} , denoted $[\mathbf{w}]$, by $[\mathbf{w}] = \{w_i \neq \# : i = 1, 2, \dots, r\}$. That is, $[\mathbf{w}]$ consists of all symbols of Σ which appear in \mathbf{w} .

Definition 1. A finite-state unification based automaton (over Σ) or, shortly, FSUBA, is a system $\mathbf{A} = (\Sigma, Q, q_0, F, \mathbf{u}, \Theta, \mu)$, where

- $Q, q_0 \in Q$, and $F \subseteq Q$ are a finite set of states, the initial state, and the set of final states, respectively.
- $\mathbf{u} = u_1u_2 \cdots u_r \in (\Sigma \cup \{\#\})^r$, $r \geq 1$, is the initial assignment – register initialization: the symbol in the i th register is u_i . Recall that $\#$ is reserved to denote an empty register. That is, if $u_i = \#$, then the i th register is empty.
- $\Theta \subseteq [\mathbf{u}]$ is the “read only” alphabet whose symbols cannot be copied into empty registers¹. One may think of Θ as a set of the language constants which cannot be unified, cf [7].

¹ Of course, we could let Θ be any subset of Σ . However, since the elements of Θ cannot be copied into empty registers, the automaton can make a move with the input from Θ only if the symbol already appears in one of the automaton registers, i.e., belongs to the initial assignment.

- $\mu \subseteq Q \times \{1, 2, \dots, r\} \times 2^{\{1, 2, \dots, r\}} \times Q$ is the transition relation whose elements are called transitions. The intuitive meaning of μ is as follows. If the automaton is in state q reading the symbol σ and there is a transition $(q, m, S, q') \in \mu$ such that the m th register either contains σ or is empty, then the automaton can enter state q' , write σ in the m th register (if it is empty), and erase the content of the registers whose indices belong to S . The m th register will be referred to as the transition register.

Like in the case of FSDA, an actual state of \mathbf{A} is a state of Q together with the contents of all its registers. That is, \mathbf{A} has infinitely many states which are pairs (q, \mathbf{w}) , where $q \in Q$ and \mathbf{w} is a word of length r – the content of the registers of \mathbf{A} . These pairs are called *configurations* of \mathbf{A} . The set of all configurations of \mathbf{A} is denoted Q^c . The pair (q_0, \mathbf{u}) , denoted q_0^c , is called the *initial configuration*², and the configurations with the first component in F are called *final configurations*. The set of final configurations is denoted F^c .

Transition relation μ induces the following relation μ^c on $Q^c \times \Sigma \times Q^c$.

Let $q, q' \in Q$, $\mathbf{w} = w_1 w_2 \dots w_r$ and $\mathbf{w}' = w'_1 w'_2 \dots w'_r$. Then the triple $((q, \mathbf{w}), \sigma, (q', \mathbf{w}'))$ belongs to μ^c if and only if there is a transition $(q, m, S, q') \in \mu$ such that the following conditions are satisfied.

- Either $w_m = \#$ (i.e., the transition register is empty in which case σ is copied into it) and $\sigma \notin \Theta$, or $w_m = \sigma$ (i.e., the transition register contains σ).
- If $m \notin S$, then $w'_m = \sigma$, i.e., if the transition register is not reset in the transition, its content is σ .
- For all $j \in S$, $w'_j = \#$.
- For all $j \notin S \cup \{m\}$, $w'_j = w_j$.

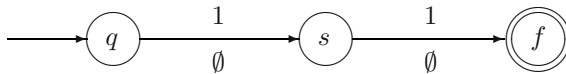
Let $\sigma = \sigma_1 \sigma_2 \dots \sigma_n$ be a word over Σ . A *run* of \mathbf{A} on σ consists of a sequence of configurations c_0, c_1, \dots, c_n such that c_0 is the initial configuration q_0^c and $(c_{i-1}, \sigma_i, c_i) \in \mu^c$, $i = 1, 2, \dots, n$.

We say that \mathbf{A} *accepts* σ , if there exists a run c_0, c_1, \dots, c_n of \mathbf{A} on σ such that $c_n \in F^c$. The set of all words accepted by \mathbf{A} is denoted by $L(\mathbf{A})$ and is referred to as an FSUBA-language.

Example 1. ([8]) Let $\mathbf{A} = (\Sigma, \{q, s, f\}, q, \{f\}, \#, \emptyset, \mu)$ be a one-register FSUBA, where μ consists of the following two transitions:

- $(q, 1, \emptyset, s)$
- $(s, 1, \emptyset, f)$.

Alternatively, μ can be described by the following diagram.



² Recall that q_0 and \mathbf{u} denote the initial state and the initial assignment, respectively.

It can be easily seen that $L(\mathbf{A}) = \{\sigma_1\sigma_2 \in \Sigma^2 : \sigma_1 = \sigma_2\}$: an accepting run of \mathbf{A} on the word $\sigma\sigma$ is $(q, \#), (s, \sigma), (f, \sigma)$.

In contrast, the language $L = \{\sigma_1\sigma_2 \in \Sigma^2 : \sigma_1 \neq \sigma_2\}$ is not an FSUBA language³. To prove that, assume to the contrary that for some FSUBA $\mathbf{A} = (\Sigma, Q, q_0, F, \mathbf{u}, \Theta, \mu)$, $L = L(\mathbf{A})$. Since Σ is infinite and $[\mathbf{u}]$ is finite, $\Sigma \setminus [\mathbf{u}]$ contains two different symbols σ_1 and σ_2 . By the definition of L , it contains the word $\sigma_1\sigma_2$. Let $(q_0, \mathbf{u}), (q_1, \mathbf{w}_1), (q_2, \mathbf{w}_2)$, $\mathbf{w}_i = w_{i,1}w_{i,2} \cdots w_{i,r}$, $i = 1, 2$, be an accepting run of \mathbf{A} on $\sigma_1\sigma_2$ and let m be the transition register between configurations (q_1, \mathbf{w}_1) and (q_2, \mathbf{w}_2) . Since neither of σ_1 and σ_2 belongs to \mathbf{u} and $\sigma_1 \neq \sigma_2$, $w_{1,m} = \#$ and $w_{2,m} = \sigma_2$. Then, replacing $w_{2,m}$ with σ_1 in $(q_0, \mathbf{u}), (q_1, \mathbf{w}_1), (q_2, \mathbf{w}_2)$ we obtain an accepting run of \mathbf{A} on $\sigma_1\sigma_1$ which contradicts $L = L(\mathbf{A})$ ⁴.

Example 2. ([8]) Let $\mathbf{A} = (\Sigma, Q, q_0, F, \mathbf{u}, \Theta, \mu)$ be an FSUBA such that $\#$ does not appear in $[\mathbf{u}]$ and for all $(q, m, S, q') \in \mu$, $S = \emptyset$. Then $L(\mathbf{A})$ is a regular language over $[\mathbf{u}]$. In general, since the restriction of a set of configurations to a finite alphabet is finite, the restrictions of FSUBA-languages to finite alphabets are regular, cf. [4, Proposition 1].

We conclude this section with the observation that FSUBA languages are closed under union, intersection, concatenation, and the Kleene star⁵. The proof of the closure under union and intersection⁶ is based on an alternative equivalent model of computation called FSUBA *with multiple assignment* that is similar to M-automata introduced in [4], and for the proof of the closure under concatenation and Kleene star we show that FSUBA with ϵ -transitions can be simulated by ordinary FSUBA.

3 Regular Expressions for FSUBA Languages

In this section we introduce an alternative description of FSUBA languages by the so called *unification based* expressions which are the infinite alphabet counterpart of the ordinary regular expressions.

Definition 2. Let $X = \{x_1, \dots, x_r\}$ be a set of variables such that $X \cap \Sigma = \emptyset$ and let Θ be a finite subset of Σ . Unification based regular expressions over (X, Θ) , or shortly *UB-expressions*, if (X, Θ) is understood from the context, are defined as follows.

³ It can be readily seen that L is accepted by a *finite-memory* automaton introduced in [3].

⁴ The decision procedure for the inclusion of FSUBA-languages in [8] is based on a refined version of this argument.

⁵ [8] deals with the inclusion problem only and does not address the closure properties of FSUBA languages at all.

⁶ It follows from Example 1 that FSUBA languages are not closed under complementation.

- \emptyset, ϵ ⁷, and each element of $X \cup \Theta$ are UB-expressions.
- If α_1 and α_2 are UB-expressions, then so is $(\alpha_1 + \alpha_2)$.
- If $X' \subseteq X$ and α_1 and α_2 are UB-expressions, then so is $(\alpha_1 \cdot_{X'} \alpha_2)$.
- If α_1 is a UB-expression, then so is $(\alpha_1)^*$.

Remark 1. A unification based regular expressions α over (X, Θ) can be thought of as an ordinary regular expression over (finite) alphabet $X \cup \Theta \cup \{\cdot_{X'} : X' \subseteq X\}$, that we shall also denote by α . That is, the *ordinary* regular expression α is obtained from the corresponding UB-expression by replacing its each subexpression of the form $\alpha_1 \cdot_{X'} \alpha_2$ with $\alpha_1 \cdot \cdot_{X'} \cdot \alpha_2$. We leave the formal definition to the reader.

Next we define the language generated by a UB-expression. Let α be a UB-expression over (X, Θ) and let $\mathbf{w} = w_1 w_2 \cdots w_n \in L_{X \cup \Theta \cup \{\cdot_{X'} : X' \subseteq X\}}(\alpha)$ ⁸. With the i th symbol w_i of \mathbf{w} , $i = 1, 2, \dots, n$, we associate a word $\overline{w_i} \in \Sigma \cup \{\epsilon\}$ in the following manner.

- If $w_i \in \Theta$, then $\overline{w_i} = w_i$.
- If w_i is of the form $\cdot_{X'}$, where $X' \subseteq X$, then $\overline{w_i} = \epsilon$.
- If $w_i \in X$, then $\overline{w_i}$ can be any element of $\Sigma \setminus \Theta$ such that the following condition is satisfied.

Let $w_i = x \in X$ and let j be the minimal integer greater than i such that $w_j = x$. If there is no k , $i < k < j$, such that w_k is of the form $\cdot_{X'}$ and $x \in X'$ ⁹, then $\overline{w_i} = \overline{w_j}$.

The word $\overline{\mathbf{w}} = \overline{w_1} \overline{w_2} \cdots \overline{w_n}$, where $\overline{w_i}$ is as defined above, $i = 1, 2, \dots, n$, is called an *instance* of \mathbf{w} . We denote by $L(\alpha)$ the set of all instances of all elements of $L_{X \cup \Theta \cup \{\cdot_{X'} : X' \subseteq X\}}(\alpha)$.

Example 3. Consider a UB-expression $\alpha = x \cdot_{\emptyset} (y \cdot_{\{y\}} \epsilon)^* \cdot_{\emptyset} x$ over $(\{x, y\}, \emptyset)$. It can be easily seen that $L(\alpha)$ consists of all words over Σ having the same first and last symbols.

Similarly, for a UB-expression $x \cdot_{\emptyset} x$ over $(\{x\}, \emptyset)$, $L(x \cdot_{\emptyset} x)$ is the language from Example 1.

Theorem 1. *A language is defined by a UB-expression if and only if it is accepted by an FSUBA.*

The proof of the “only if” part of the theorem is based on the closure properties of FSUBA-languages and is quite standard. The proof of the “if” part of Theorem 1 is presented in the next section.

We conclude this section with one more closure property of FSUBA-languages that is an immediate corollary to Theorem 1.

⁷ Of course, ϵ is redundant (but, still, very useful), because $L(\emptyset^*) = \{\epsilon\}$.

⁸ Here and hereafter, $L_{X \cup \Theta \cup \{\cdot_{X'} : X' \subseteq X\}}(\alpha)$ denotes the language over $X \cup \Theta \cup \{\cdot_{X'} : X' \subseteq X\}$ defined by α , see Remark 1.

⁹ That is, no subscript X' that appears between the i th and the j th positions of \mathbf{w} contains x .

Corollary 1. *FSUBA languages are closed under reversing¹⁰.*

Proof. It can be easily verified that, for a UB-expression α , $(L(\alpha))^R = L(\alpha^R)$, where α^R is defined by the following induction.

- If $\alpha \in \{\emptyset, \epsilon\} \cup X \cup \Theta$, then α^R is α .
- $(\alpha_1 + \alpha_2)^R$ is $(\alpha_1^R + \alpha_2^R)$.
- $(\alpha_1 \cdot_{X'} \alpha_2)^R$ is $(\alpha_2^R \cdot_{X'} \alpha_1^R)$.
- $((\alpha)^*)^R$ is $(\alpha^R)^*$.

4 Proof of the “if” Part of Theorem 1

The proof of the “if part” of Theorem 1 is based on a tight relationship between FSUBA and ordinary FA. We shall use the following model of FA.

Definition 3. ([5]) *A (non-deterministic) finite state automaton over a finite alphabet Σ' is a system $\mathbf{M} = (Q, q_0, F, \Delta)$, where*

- Q is a finite set of states.
- $q_0 \in Q$ is the initial state.
- $F \subseteq Q$ is the set of final states.
- Δ is a finite subset of $Q \times \Sigma'^* \times Q$ called the transition relation.

A word \mathbf{w} over alphabet Σ' is accepted by \mathbf{M} , if there is a partition $\mathbf{w} = u_1 \cdots u_n$ of \mathbf{w} , $u_i \in \Sigma'^*$, and a sequence of states s_0, s_1, \dots, s_n such that

- $s_0 = q_0$,
- $s_n \in F$, and
- and for each $i = 0, 1, \dots, n-1$, $(s_i, u_{i+1}, s_{i+1}) \in \Delta$.

Let $\mathbf{A} = (\Sigma, Q, q_0, F, \mathbf{u}, \Theta, \mu)$ be an FSUBA with r registers. By [8, Claim 2], we may assume that $[\mathbf{u}] = \Theta$. Also, changing the order of registers, if necessary, we may assume that $\mathbf{u} = \theta_1 \cdots \theta_m \#^n$, where $n + m = r$. Consider a finite state automaton $\mathbf{M}^{\mathbf{A}} = (Q, q_0, F, \Delta)$ over $\{x_1, \dots, x_n\} \cup \Theta \cup \{\cdot_{X'} \epsilon : X' \subseteq \{x_1, \dots, x_n\}\}$, where transition relation Δ is defined as follows. For every transition $(q, k, S, q') \in \mu$, Δ contains:

- $(q, \theta_k \cdot_{X'} \epsilon, q')$, if $1 \leq k \leq m$, or
- $(q, x_{(k-m)} \cdot_{X'} \epsilon, q')$, if $m+1 \leq k \leq r$,

where $X' = \{x_i : i + m \in S\}$.

Remark 2. Note that the diagrams of \mathbf{A} and $\mathbf{M}^{\mathbf{A}}$ differ only in the transition labels which can be recovered from each other in a straightforward manner.

¹⁰ It should be pointed out that FMA languages are not closed under reversing, see [4, Example 8]. The proof of the corollary shows that this constitutes rather serious obstacle to find a kind of regular expressions for FMA languages.

The proof of the “if” part of Theorem 1 is based on the fact that set of the sequences of labels of the accepting paths M^A is regular. Namely, the “if” part of Theorem 1 immediately follows from Theorem 2 below.

Theorem 2. *Let A and M^A be as above and let α be an ordinary regular expression that defines $L(M^A)$ ¹¹. Then, $L(\alpha) = L(A)$.*

Proof. (Sketch) Since $L_{\{\theta_1, \dots, \theta_m, x_1, \dots, x_n\} \cup \{\cdot_{X'} \in X' \subseteq \{x_1, \dots, x_n\}\}}(\alpha) = L(M^A)$, it suffices to show that $L(A)$ consists of all instances of the elements of $L(M^A)$.

Let $\mathbf{p} = e_1, \dots, e_n$ be a path of edges in the graph representation of A . One can think of \mathbf{p} as the diagram of an FSUBA, also denoted by \mathbf{p} . Then $L(\mathbf{p})$ consists of all words of length n over Σ which “drive A through \mathbf{p} from its first to the last vertex (state).”

Let \mathbf{P} denote the set of all paths \mathbf{p} starting from the initial state and ending at a final state. Then

$$L(A) = \bigcup_{\mathbf{p} \in \mathbf{P}} L(\mathbf{p}).$$

On the other hand, by Remark 2, \mathbf{p} has the corresponding path in M^A , also denoted by \mathbf{p} , that differs from it only in the transition labels.

The labels of \mathbf{p} in the graph representation of M^A form a UB-expression that we shall denote by $\mathbf{w}_{\mathbf{p}}$. Therefore,

$$L(M^A) = \{\mathbf{w}_{\mathbf{p}} : \mathbf{p} \in \mathbf{P}\}.$$

Consequently the equality of $L(A)$ to the set of all instances of the elements of $L(M^A)$ will follow if we prove that for each path \mathbf{p}

$$L(\mathbf{w}_{\mathbf{p}}) = L(\mathbf{p}), \tag{1}$$

i.e., $L(\mathbf{p})$ consists of all instances of $\mathbf{w}_{\mathbf{p}}$.

The proof of (1) is by induction on the length n of \mathbf{p} and is omitted.

References

1. M. Bielecki, J. Hidders, J. Paredaens, J. Tyszkiewicz, and J. Van den Bussch. Navigating with a browser. In P. Widmayer, F. Triguero, R. Morales, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *Proceedings of the 29th International Colloquium on Automata, Languages and Programming – ICALP 2002*, pages 764–775, Berlin, 2002. Springer. Lecture Notes in Computer Science 2380.
2. B. Bolling, M. Leucker, and T. Noll. Generalized regular MSA languages. Technical report, Department of Computer Science, Aachen University of Technology, 2002.
3. M. Kaminski and N. Francez. Finite-memory automata. In *Proceedings of the 31th Annual IEEE Symposium on Foundations of Computer Science*, pages 683–688, Los Alamitos, CA, 1990. IEEE Computer Society Press.
4. M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science A*, 138:329–363, 1994.

¹¹ Recall that, by Remark 1, α is a UB-expression as well.

5. H.R. Lewis and C.H. Papadimitriou. *Elements of the theory of computation*. Prentice-Hall, Inc., Englewood Cliffs, NJ, USA, 1981.
6. F. Neven, T. Schwentik, and V. Vianu. Towards regular languages over infinite alphabets. In J. Sgall, A. Pultr, and P. Kolman, editors, *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science*, pages 560–572, Berlin, 2001. Springer. Lecture Notes in Computer Science 2136.
7. Y. Shemesh and N. Francez. Finite-state unification automata and relational languages. *Information and Computation*, 114:192–213, 1994.
8. A. Tal. Decidability of inclusion for unification based automata. M.Sc. thesis, Department of Computer Science, Technion – Israel Institute of Technology, 1999.

On the Power of One-Sided Error Quantum Pushdown Automata with Classical Stack Operations

Masaki Nakanishi

Graduate School of Information Science, Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara 630-0101, Japan
`m-naka@is.naist.jp`

Abstract. One of important questions on quantum computing is whether there is a computational gap between the models that are allowed to use quantum effects and the models that are not. Several types of quantum computation models have been proposed, including quantum finite automata and quantum pushdown automata (with quantum pushdown stack). It has been shown that some quantum computation models are more powerful than classical counterparts and some are not since quantum computation models are required to obey some restrictions such as reversible state transitions. In this paper, we investigate the power of quantum pushdown automata whose stack is assumed to be implemented as a classical device, and show that they are strictly more powerful than classical counterparts in the one-sided error setting. That is, we show that there is a non-context-free language which quantum pushdown automata with classical stack operations can recognize with one-sided error.

1 Introduction

One of important questions on quantum computing is whether there is a computational gap between the models that are allowed to use quantum effects and the models that are not. Several types of quantum computation models have been proposed, including quantum finite automata and quantum pushdown automata [1–4]. In [2], it is shown that quantum pushdown automata can recognize every regular language and some non-context-free languages. However, it is still open whether quantum pushdown automata are more powerful than probabilistic pushdown automata or not. The quantum pushdown automaton model introduced in [2] has a quantum tape head and a quantum stack, and needs $O(n)$ qubits for realization, where n is the execution time.

In [4], we introduced another model of quantum pushdown automata, called quantum pushdown automata with classical stack operations (QCPDA), whose stack is assumed to be implemented as a classical device. This means that a QCPDA needs $\lceil \log m \rceil$ qubits for specifying the position of the head, where m is an input length, in addition to the constant number of qubits for representing the finite state control. And we showed that QCPDA's can simulate any

(even non-reversible) probabilistic pushdown automata with the same acceptance probability as that of the original probabilistic pushdown automata in [4]. This implies that QCPDA's are at least as powerful as probabilistic pushdown automata. However, it remains open whether QCPDA's are *strictly* more powerful than classical counterparts, i.e., whether QCPDA's can recognize some languages which classical counterparts cannot recognize.

In this paper, we show that QCPDA's are strictly more powerful than classical counterparts in the one-sided error setting. That is, we show that there is a language which a QCPDA can recognize with one-sided error. We also show that this language cannot be recognized by non-deterministic pushdown automata, that is, it is not a context-free language. This implies that it cannot be recognized by probabilistic pushdown automata with one-sided error, either.

This paper is organized as follows: In Sect. 2, we define quantum pushdown automata with classical stack operations and probabilistic pushdown automata. In Sect. 3, we show that quantum pushdown automata with classical stack operations can recognize a certain language L_1 with one-sided error. We also show that L_1 is not a context-free language. Sect. 4 concludes this paper.

2 Preliminaries

In this section, we give a description of quantum pushdown automata with classical stack operations (QCPDA) defined in [4]. A QCPDA has an input tape to which a quantum head is attached, and has a classical stack to which a classical stack top pointer is attached. A QCPDA has a quantum state control. The quantum state control reads the stack top symbol pointed by the classical stack top pointer and reads the input symbol pointed by the quantum head. A stack operation is determined solely by the result of an observation of the quantum state control. We define QCPDA's formally as follows.

Definition 1. *A Quantum Pushdown Automaton with Classical Stack Operations (QCPDA) is defined as the following 8-tuple:*

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \sigma, Q_{acc}, Q_{rej}),$$

where Q is a set of states, Σ is a set of input symbols including the left and the right end-marker $\{\$, \$\}$ respectively, Γ is a set of stack symbols including the bottom symbol Z , δ is a quantum state transition function ($\delta : (Q \times \Sigma \times \Gamma \times Q \times \{0, 1\}) \longrightarrow \mathbb{C}$), q_0 is an initial state, σ is a function by which stack operations are determined ($\sigma : Q \setminus (Q_{acc} \cup Q_{rej}) \longrightarrow \Gamma^+ \cup \{-, pop\}$), $Q_{acc} (\subseteq Q)$ is a set of accepting states, and $Q_{rej} (\subseteq Q)$ is a set of rejecting states, where $Q_{acc} \cap Q_{rej} = \emptyset$. \square

$\delta(q, a, b, q', D) = \alpha$ means that the amplitude of the transition from q to q' moving the quantum head to D ($D = 1$ means 'right' and $D = 0$ means 'stay') is α when reading an input symbol a and a stack symbol b . A configuration of the quantum portion of a QCPDA is a pair (q, k) , where k is the position of the

quantum head and q is in Q . It is obvious that the number of configurations of the quantum portion is $n|Q|$, where n is an input length.

A superposition of configurations of the quantum portion of a QCPDA is any element of $l_2(Q \times \mathbb{Z}_n)$ of unit length, where $\mathbb{Z}_n = \{0, 1, \dots, n-1\}$. For each configuration, we define a column vector $|q, k\rangle$ as follows:

- $|q, k\rangle$ is an $n|Q| \times 1$ column vector.
- The row corresponding to (q, k) is 1, and the other rows are 0.

For an input word \mathbf{x} (i.e., the string on the input tape between \pounds and $\$$) and a stack symbol a , we define a time evolution operator $U_a^{\mathbf{x}}$ as follows:

$$U_a^{\mathbf{x}}(|q, k\rangle) = \sum_{q' \in Q, D \in \{0,1\}} \delta(q, x(k), a, q', D) |q', k + D\rangle,$$

where $x(k)$ is the k -th input symbol of the input \mathbf{x} . If $U_a^{\mathbf{x}}$ is unitary (for any $a \in \Gamma$ and for any input word \mathbf{x}), that is, $U_a^{\mathbf{x}} U_a^{\mathbf{x}\dagger} = U_a^{\mathbf{x}\dagger} U_a^{\mathbf{x}} = I$, where $U_a^{\mathbf{x}\dagger}$ is the transpose conjugate of $U_a^{\mathbf{x}}$, then we say that the corresponding QCPDA is well-formed. A well-formed QCPDA is considered to be valid in terms of the quantum theory. We consider only well-formed QCPDA's.

We define the notion of “words accepted by a QCPDA” in the following. For that purpose, we first describe the way how the quantum portion and the classical stack of a QCPDA work.

Let the initial quantum state and the initial position of the head be q_0 and ‘0’ respectively. We define as $|\psi_0\rangle = |q_0, 0\rangle$. We also define as follows:

$$\begin{aligned} E_w &= \text{span}\{|q, k\rangle \mid \sigma(q) = w\}, \\ E_{\text{acc}} &= \text{span}\{|q, k\rangle \mid q \in Q_{\text{acc}}\}, \\ E_{\text{rej}} &= \text{span}\{|q, k\rangle \mid q \in Q_{\text{rej}}\}. \end{aligned}$$

We define the observable \mathcal{O} as $\mathcal{O} = \oplus_j E_j$, where j is $w \in \Gamma^+ \cup \{-, \text{pop}\}$, ‘acc’, or ‘rej’. For notational simplicity, we define the outcome of an observation corresponding to E_j to be j .

The computation of the QCPDA proceeds as follows:

For an input word \mathbf{x} , the quantum portion works as follows:

- (a) $U_a^{\mathbf{x}}$ is applied to $|\psi_i\rangle$. Let $|\psi_{i+1}\rangle = U_a^{\mathbf{x}} |\psi_i\rangle$, where a is the stack top symbol.
- (b) $|\psi_{i+1}\rangle$ is observed with respect to the observable $\mathcal{O} = \oplus_j E_j$. Let $|\phi_j\rangle$ be the projection of $|\psi_{i+1}\rangle$ to E_j . Then each outcome j is obtained with probability $|\langle \phi_j | \psi_{i+1} \rangle|^2$. Note that this observation causes $|\psi_{i+1}\rangle$ to collapse to $\frac{1}{\|\phi_j\|} |\phi_j\rangle$, where j is the obtained outcome. Then go to (c).

The classical stack works as follows:

- (c) Let the outcome of the observation be j . If j is ‘acc’ (‘rej’, resp.) then it outputs ‘accept’ (‘reject’, resp.), and the computation halts. If j is ‘-’, then the stack is unchanged. If j is ‘pop’, then the stack top symbol is popped. Otherwise (j is a word in Γ^+ in this case), the word j is pushed. Then, go to (a) and repeat.

We call the above (a), (b) and (c) ‘one step’ collectively.

For a language L , if there exists a constant ε ($0 \leq \varepsilon < 1$) that does not depend on inputs, a QCPDA accepts any word in L with probability greater than $1 - \varepsilon$, and it rejects any word which is not in L with certainty, then we say that L is recognized by the QCPDA with one-sided error.

To check well-formedness, the following theorem is shown in [4].

Theorem 1. *A QCPDA is well-formed if the quantum state transition function of M satisfies the following conditions for any $a \in \Gamma$:*

$$\begin{aligned} (1) \quad & \forall q_1, q_2, b \\ & \sum_{q', D} \overline{\delta(q_1, b, a, q', D)} \delta(q_2, b, a, q', D) = \begin{cases} 1 & (q_1 = q_2) \\ 0 & (q_1 \neq q_2), \end{cases} \\ (2) \quad & \forall q_1, q_2, b_1, b_2 \\ & \sum_{q'} \overline{\delta(q_1, b_1, a, q', 0)} \delta(q_2, b_2, a, q', 1) = 0, \end{aligned}$$

where $\overline{\delta(\cdot, \cdot, \cdot, \cdot)}$ is the conjugate of $\delta(\cdot, \cdot, \cdot, \cdot)$. □

For simplicity, we will handle only a subclass of QCPDA's, called *simplified* QCPDA's, such that we can decompose the quantum state transition function into two functions: one for changing states and the other for moving the quantum head. For $a \in \Sigma$ and $b \in \Gamma$, we adopt a linear operator $V_{a,b} : l_2(Q) \rightarrow l_2(Q)$ for changing states, and a function $\Delta : Q \rightarrow \{0, 1\}$ for moving the quantum head. In *simplified* QCPDA's, direction of the movement of the head is determined solely by the state to which the current state makes a transition. Then the transition function δ we are concerned with is described as follows:

$$\delta(q, a, b, q', D) = \begin{cases} \langle q' | V_{a,b} | q \rangle & (\Delta(q') = D) \\ 0 & (\Delta(q') \neq D), \end{cases}$$

where $\langle q' | V_{a,b} | q \rangle$ is the coefficient of $|q'\rangle$ in $V_{a,b}|q\rangle$. The following theorem is also shown in [4].

Theorem 2. *A simplified QCPDA is well-formed if, for any $a \in \Sigma$ and $b \in \Gamma$, the linear operator $V_{a,b}$ satisfies the following condition:*

$$\sum_{q'} \overline{\langle q' | V_{a,b} | q_1 \rangle} \langle q' | V_{a,b} | q_2 \rangle = \begin{cases} 1 & (q_1 = q_2) \\ 0 & (q_1 \neq q_2) \end{cases}$$

□

We define probabilistic pushdown automata as follows.

Definition 2. *A Probabilistic Pushdown Automaton (PPA) is defined as the following 7-tuple:*

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Q_{acc}, Q_{rej}),$$

where Q is a set of states, Σ is a set of input symbols including the left and the right end-marker $\{\$, \$\}$ respectively, Γ is a set of stack symbols including the

bottom symbol Z , δ is a state transition function ($\delta : (Q \times \Sigma \times \Gamma \times Q \times \{0, 1\}) \times \Gamma^+ \cup \{-, \text{pop}\} \rightarrow [0, 1]$), q_0 is an initial state, $Q_{\text{acc}}(\subseteq Q)$ is a set of accepting states, and $Q_{\text{rej}}(\subseteq Q)$ is a set of rejecting states, where $Q_{\text{acc}} \cap Q_{\text{rej}} = \emptyset$. \square

$\delta(q, a, b, q', D, w) = \alpha$ means that the probability of the transition from q to q' moving the head to D with the stack operation w is α when reading an input symbol a and a stack symbol b . Note that the sum of the weights (i.e. the probabilities) of outgoing transitions of a state must be 1. Computation halts when it enters the accepting or rejecting states.

In [4], it is shown that QCPDA's can simulate any PPA's with the same acceptance probability as that of the original PPA's.

Theorem 3. *A QCPDA can simulate any PPA with the same acceptance probability as that of the original PPA.* \square

We give a sketch of the proof.

Sketch of Proof. We say a PPA to be of post-state-dependent type if the stack operation and the direction of the head movement are determined solely by the next state rather than by the triple of (current) state, input symbol and stack-top symbol, where the next state is the state after transition.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Q_{\text{acc}}, Q_{\text{rej}})$ be a PPA. It is straightforward to see that any PPA can be converted to a PPA of post-state-dependent type. We assume that M is of post-state-dependent type without loss of generality.

We make M into a reversible PPA by adding extra states and stack symbols, where *reversible* means that each state has at most one incoming transition for each stack symbol. Suppose that a state q has multiple incoming transitions for a stack symbol a , that is, there are at least two distinct transitions, $\delta(q_1, x_1, a, q, D, w) = p_1$ ($p_1 \neq 0$) and $\delta(q_2, x_2, a, q, D, w) = p_2$ ($p_2 \neq 0$). We consider the case of $w \in \Gamma^+$. The other cases are similar. For each such incoming transition $t : \delta(q', x, a, q, D, w) = p$ ($p \neq 0$), we define a new state q_t . We add a new stack symbol s_t to Γ and define δ as $\delta(q', x, a, q_t, 0, ws_t) = p$ and $\delta(q_t, x, s_t, q, D, \text{pop}) = 1$, where ws_t is a concatenation of w and s_t . Then we remove the original transition, that is, we define as $\delta(q', x, a, q, D, w) = 0$. We illustrate an example of how to add new states in Fig. 1. The revised PPA is reversible. Note that the revised PPA is also of post-state-dependent type. Since each state has at most one incoming transition for each stack symbol, it is straight forward to see that the revised reversible PPA can be converted to a QCPDA, that is, the corresponding time evolution operator $U_a^{\mathbf{x}}$ can be unitary by defining thus far undefined transitions properly. \square

3 On the Power of QCPDA's with One-Sided Error

In this section, we show that QCPDA's can recognize the following language L_1 with one-sided error while probabilistic pushdown automata cannot with one-sided error.

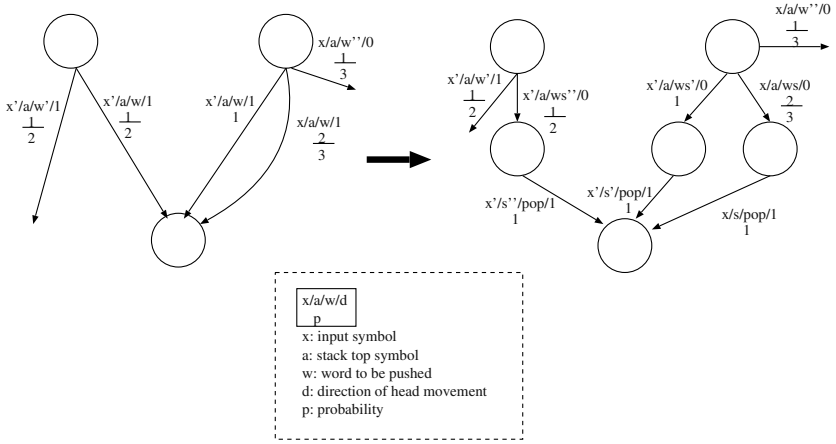


Fig. 1. How to add new states.

$$L_1 = \left\{ u \# v \natural w b x \% y \mid \begin{array}{l} u, v, w, x, y \in \{a, b\}^*, \\ \neg \left(|u| = |v| = |w| = |x| \right) \\ \text{and } v = w \\ \text{and } \neg \left(|u| = |x| \text{ and } |v| = |w| = 0 \right) \\ \text{and } (y = v^R \text{ or } y = w^R) \end{array} \right\}$$

Theorem 4. QCPDA's can recognize L_1 with one-sided error.

Proof. We show a QCPDA M that recognizes L_1 . The outline of M is as follows: M rejects any input which is not of the form $u \# v \natural w b x \% y = \{a, b\}^* \# \{a, b\}^* \natural \{a, b\}^* b \{a, b\}^* \% \{a, b\}^*$. M consists of the three components M_1 , M_2 and M_3 , each of which is a deterministic automaton. M_1 processes the substring $u \# v \natural w b x$ as follows:

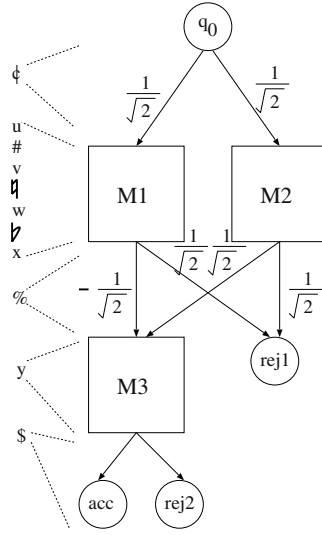
1. M_1 reads u taking $2|u|$ steps with the stack unchanged,
2. pushes each symbol of v one by one,
3. and reads w and x taking $|w| + |x|$ steps with the stack unchanged.

M_2 processes the substring $u \# v \natural w b x$ as follows:

1. M_2 reads u and v taking $|u| + |v|$ steps with the stack unchanged,
2. pushes each symbol of w one by one,
3. and reads x taking $2|x|$ steps with the stack unchanged.

M_3 processes the substring y checking whether the string on the stack is y^R or not.

M_1 and M_2 process the substring $u \# v \natural w b x \%$ in parallel using superposition. Firstly, we consider the case that $(|u| = |v| = |w| = |x| \text{ and } v = w)$ or $(|u| = |x| \text{ and } |v| = |w| = 0)$. In this case, M_1 and M_2 push the same symbol at the same

Fig. 2. QCPDA M .

time, and also read ‘%’ at the same time. Thus, when reading ‘%’, M_1 and M_2 interfere with each other and move to the rejecting state with certainty. (See Fig. 2).

Secondly, we consider the case that $\neg(|u| = |v| = |w| = |x| \text{ and } v = w)$ and $\neg(|u| = |x| \text{ and } |v| = |w| = 0)$. In this case, the stack operations are different between M_1 and M_2 at some time. Then the observation, the result of which corresponds to the stack operation, causes the superposition to collapse to either M_1 or M_2 . Thus M moves to the third component M_3 with probability $1/2$ with v or w on the stack. Thus if $v = y^R$ or $w = y^R$, M accepts the input with probability $1/4$.

We define M in detail in the following.

M has the following states:

- $q_{1,u,1,-}, q_{1,u,0,-}, q_{1,\#,0,-}, q_{1,v,1,-}, q_{1,v,0,a}, q_{1,v,0,b}, q_{1,w,1,-}, q_{1,x,1,-}$.
- $q_{2,u,1,-}, q_{2,v,1,-}, q_{2,w,1,-}, q_{2,w,0,a}, q_{2,w,0,b}, q_{2,x,1,-}, q_{2,x,0,-}, q_{2,\%,0,-}$.
- $q_{3,y,1,-}, q_{3,y,0,pop}$.
- $q_0, q_{acc}, q_{rej1}, q_{rej2}, q_{rej3}, q_{rej4}, q_{rej5}, q_{rej6}, q_{rej7}, q_{rej8}, q_{rej9}, q_{rej10}, q_{rej11}$.

q_{acc} is an accepting state. $q_{rej1}, \dots, q_{rej11}$ are rejecting states. The index of $q_{i,z,d,c}$ means that the state is used to process a substring z in M_i , $\Delta(q_{i,z,d,c}) = d$, and c represents the value of $\sigma(q_{i,z,d,c})$ as follows: the stack top symbol is popped ($c = \text{‘pop’}$), the stack is unchanged ($c = -$), and c is pushed (otherwise). The initial state is q_0 .

The state transition diagrams of the components M_1 , M_2 and M_3 are illustrated in Fig. 3. It is straightforward to see that the corresponding time evolution operators can be extended to be unitary by determining thus far undefined transitions properly by Theorem 2. \square

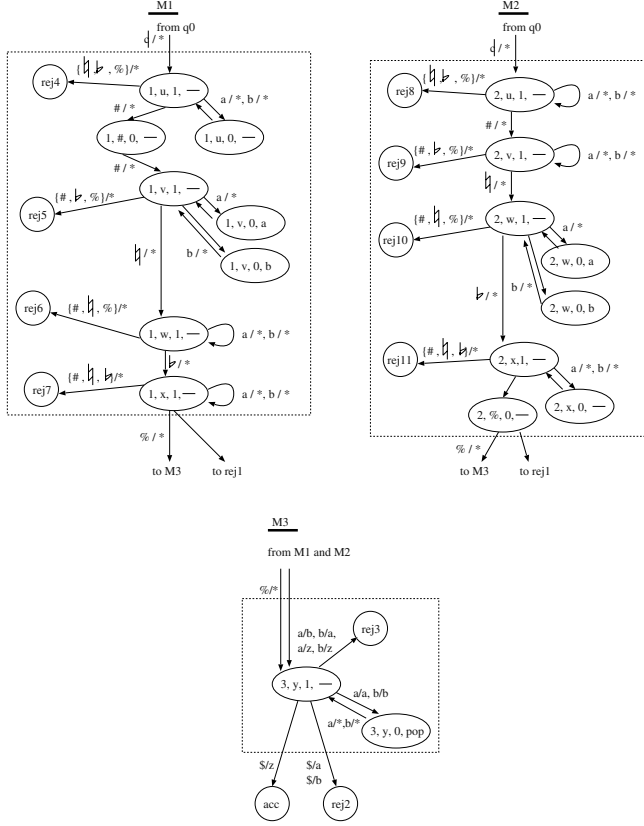


Fig. 3. M_1 , M_2 and M_3 . Each edge label represents a pair of an input symbol and a stack top symbol, where ‘*’ is a wild card.

We show that a non-deterministic pushdown automaton cannot recognize L_1 , that is, L_1 is not a context-free language. It is straight forward to see that this means that a probabilistic pushdown automaton cannot recognize L_1 with one-sided error.

Theorem 5. *A non-deterministic pushdown automaton cannot recognize L_1 .* □

We use Ogden’s lemma [5] to prove Theorem 5.

Ogden’s Lemma. *For any context-free language L , $\exists n \in \mathbb{N}$, such that $\forall z \in L$, if d positions in z are “distinguished,” with $d > n$, then $\exists u, v, w, x, y$ such that $z = uvwx^i y$ and*

1. vx contains at least one distinguished position;
 2. if r is the number of distinguished positions in vw , then $r \leq n$;
 3. $\forall i \in \mathbb{N}$, $uv^iwx^i y \in L$.
-

Proof of Theorem 5. Suppose that L_1 were a context-free language. We consider a word $z = p\#q\sharp r\flat s\%t = a^{n!+n}\#a^n\sharp a^{n!+n}ba^{n!+n}\%a^n \in L_1$. Let each symbol in q be distinguished. Then $z = uvwxy$ satisfies the conditions of Ogden's lemma. If either v or x contains a separator ($\#, \sharp, \flat, \%$), then $z' = uv^2wx^2y \notin L_1$. Note that vx contains at least one 'a' in q . Thus x must contain 'a's in t . (Otherwise $z' = uv^2wx^2y (= p'\#q'\sharp r'\flat s'\%t') \notin L_1$ since $q' \neq t'^R$ and $r' \neq t'^R$). We assume that v contains c 'a's and $cd = n!$. Then $uv^{d+1}wx^{d+1}y (= p''\#q''\sharp r''\flat s''\%t'') \notin L_1$ since $|p''| = |q''| = |r''| = |s''|$ and $q'' = r''$, a contradiction. Thus L_1 is not a context-free language. \square

By Theorem 3, 4, and 5, we can conclude that one-sided error QCPDA's are strictly more powerful than one-sided error probabilistic pushdown automata.

4 Conclusion

We have shown that QCPDA's can recognize a non-context-free language with one-sided error. Since the class of languages recognized by one-sided error PPA's is contained in the class of non-context-free languages, L_1 cannot be recognized by a one-sided error PPA. Thus, QCPDA's are strictly more powerful than PPA's in the one-sided error setting. Our conjecture is that L_1 cannot be recognized by probabilistic pushdown automata even with two-sided error, which implies that QCPDA's (with two-sided error) are strictly more powerful than probabilistic pushdown automata.

Acknowledgement

We thank Shigeru Yamashita for helpful discussions. This work was supported in part by Grants-in-Aid for Scientific Research (no. 15700014) from the Ministry of Education, Culture, Sports, Science and Technology of Japan and by an Okawa Foundation Research Grant.

References

1. A. Ambainis and J. Watrous, "Two-way finite automata with quantum and classical states," *Theoretical Computer Science*, vol. 287, no. 1, pp.299–311, 2002.
2. M. Golovkins, "Quantum Pushdown Automata," *Proc. of 27th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM2000)*, LNCS 1963, pp. 336–346, 2000.
3. C. Moore and J. P. Crutchfield, "Quantum automata and quantum grammars," *Theoretical Computer Science*, 237(1–2):275–306, 2000.
4. M. Nakanishi, T. Indoh, K. Hamaguchi, and T. Kashiwabara, "Expressive Power of Quantum Pushdown Automata with a Classical Stack," *Booklet of 3rd International Conference on Unconventional Models of Computation (UMC'02)*, October 2002.
5. W. Ogden, "A Helpful Result for Proving inherent Ambiguity," *Math. Syst. Theory* 2, pp.191 – 194, 1968.

Learning DNFs and Circuits Using Teaching Assistants

N. Variyam Vinodchandran

Department of Computer Science and Engineering
University of Nebraska-Lincoln
vinod@cse.unl.edu

Abstract. In this paper we consider the complexity of learning monotone DNF formulae and Boolean circuits in an exact learning model called the *teaching assistant model*. This model, introduced in one of our earlier works, uses teaching assistant classes for classifying learning problems. Teaching assistant classes are a learning-theoretic analog of complexity classes. We show that monotone DNFs are learnable using a teaching assistant in the class SPP. On the other hand, Boolean circuits and general DNFs are not learnable using an SPP teaching assistant unless $\text{NP} \subseteq \text{SPP}$. We also show that learnability of Boolean circuits with an assistant in the class NP will improve the Karp-Lipton collapse result to P^{NP} .

1 Introduction

In this paper we study the complexity of learning certain concept classes in an exact learning model called the *teaching assistant* model. The teaching assistant model, developed in our earlier work [3, 4], is a refinement of Angluin's well-known exact learning model. In Angluin's model a learner learns a target concept by making either membership queries or equivalence queries (or both) to an honest teacher. In the teaching assistant model, a complexity-theoretic approach is taken towards learning.

The teaching assistant model: A learning model based on complexity classes. A central idea in this model is the notion of a *teaching assistant*, an intermediate agent between the learner and the teacher. The learner, instead of querying the teacher directly, queries a teaching assistant to learn the target concept. The teaching assistant in turn makes membership queries to the teacher to gather information about the target concept. The usefulness of this model for investigating the complexity of learning problems comes from the fact that we can classify teaching assistants into *teaching assistant classes*, which is similar to the way we classify computational problems into complexity classes.

This complexity-theoretic model for learning is closely related to Angluin's exact learning model. An equivalence query to a teacher can be replaced by a sequence of queries to an NP oracle, which in turn makes membership queries to the teacher. In particular, an NP-like set $\{\langle y, r \rangle \mid y \text{ is a prefix of a counter example to the hypothesis } r\}$ can be used to prefix-search for a counter example for an

equivalence query r . This NP oracle is an example of a teaching assistant. Hence, we can say that classes learnable with equivalence queries in Angluin's model are also learnable in the teaching assistant model with an NP teaching assistant. Similarly, a membership query can be seen as a query to a teaching assistant in the class P. Therefore, any concept classes learnable with only membership queries in Angluin's model are also learnable with a P teaching assistant.

The main advantage of restating the learnability using queries in the language of NP and P is that we can generalize this idea to other complexity classes. For example, consider the complexity class UP. UP is the class of decision problems accepted by nondeterministic machines with a restriction that there is at most one accepting computation for any instance of the problem. The complexity of problems in UP is believed to be strictly between P and NP. Hence we can say that the concept classes that are learnable using a teaching assistant in the class UP have less learning complexity than concept classes that require assistants from the class NP. In a certain well-defined sense, the complexity of teaching assistants quantify the complexity of learning. In general, we can define a teaching assistant class analogous to any complexity class. Hence, introducing this intermediary called the teaching assistant in the exact learning model provides a framework which is suitable for analyzing the complexity of learning problems. We call this framework the teaching assistant model (TA model) for exact learning.

In this paper we are interested in the teaching assistant classes analogous to the complexity classes NP and SPP. While NP is a well-known class, some motivation needs to be given for studying the class SPP.

The complexity class SPP and its significance. SPP was introduced and studied by Fenner, Fortnow, and Kurtz in [8]. This class was also independently studied in [14] under the name XP and in [9] under the name ZUP). SPP consists of decision problems that are accepted by nondeterministic polynomial time Turing machines such that; for positive instances, the difference in the number of accepting and rejecting paths of the machine is exactly one, and for negative instances, this difference is zero. SPP can be seen as the *gap* (difference between accepting and rejecting paths) analog of the class UP.

From a structural point of view, this class is interesting mainly because of its *lowness* property. A complexity class C_1 is said to be *low* for another (relativizable) complexity class C_2 if $C_2^{C_1} = C_2$. That is, any C_2 -machine that uses a language in C_1 as an oracle can be converted into another C_2 -machine with the same acceptance behavior without any oracle access: C_1 is powerless as oracle to C_2 . It is shown in [8] that SPP is low for counting complexity classes such as PP, $C=P$, and $\text{Mod}_k P$ for $k \geq 2$. Thus, intuitively, SPP is a class of "low counting complexity".

Although SPP is a structurally defined class, it has been proved to be very useful in analyzing the complexity of "intermediate problems"—problems that are not known to be in P, but unlikely to be NP-complete. Membership of a problem in the class SPP can be seen as evidence that the problem is unlikely to be NP-complete: no known NP-complete problem has been shown to be in

SPP. In [13,15] it is shown that many of the known candidate problems of intermediate complexity from the domain of algebra are in the class SPP. Very recently, Arvind and Kurur [2] presented an elegant SPP algorithm for the well-known Graph Isomorphism problem.

The exact power of SPP in relation to NP is not yet clear. Because of the strong restrictions on the computation tree of nondeterministic Turing machines accepting SPP languages, it seems surprising that NP could be in SPP. Furthermore, there are oracles separating NP from SPP [5]. On the other hand, results from derandomization and resource bounded measure theory give some evidence towards the containment of NP in SPP [12,11].

Prior results on teaching assistant model. In [3], the teaching assistant framework was introduced primarily to distinguish the complexity of learning certain algebraic concepts such as vector spaces and permutation groups from the nonalgebraic concept class CNF. In particular, it was shown that vector spaces can be learned using teaching assistants in the class SPP. Learning CNFs using a teaching assistant in the class SPP is not possible, unless $\text{NP} \subseteq \text{SPP}$. A subsequent work initiated a study of teaching assistant classes analogous to many well-studied complexity classes such as P, UP, NP, $(\text{UP} \cap \text{co-UP})$, $(\text{NP} \cap \text{co-NP})$, Σ_2^P , SPP, LWPP, and established their importance in classifying the complexity of learning [4]. In particular, [4] gave a fine separation among many of the teaching assistant classes mentioned above. For example, this paper presented a concept class that is learnable using a $(\text{UP} \cap \text{coUP})$ teaching assistant, but not learnable with teaching assistants in P. Note that, unlike in complexity theory where establishing proper separations among complexity classes is extremely difficult, we can actually show separations among TA classes.

New results in this paper. In this paper, we prove new results on the complexity of learning monotone DNFs and Boolean circuits in the TA model. First, we show that monotone DNFs are learnable using teaching assistants in the class SPP. On the other hand, general DNFs and Boolean circuits are not learnable using teaching assistants in the class SPP unless $\text{NP} \subseteq \text{SPP}$. We also show that if Boolean Circuits are learnable with an NP assistant then we can improve the current best upper bound of the Karp-Lipton collapse result; under this assumption we show that $\text{SAT} \in \text{P/poly} \Rightarrow \text{PH} \subseteq \text{P}^{\text{NP}}$. The current best upper bound is S_2^P ; a complexity class that contains P^{NP} . Previously, it was known that $\text{SAT} \in \text{P/poly} \Rightarrow \text{PH} \subseteq \text{ZPP}^{\text{NP}}$ (attributed to Watanabe in [6]). More recently, Cai showed that $\text{S}_2^P \subseteq \text{ZPP}^{\text{NP}}$ [7]. Cai's result together with the result that $\text{SAT} \in \text{P/poly} \Rightarrow \text{PH} \subseteq \text{S}_2^P$ (attributed to Sengupta in [7]) improved the Karp-Lipton upper bound from ZPP^{NP} to S_2^P . There are relativized worlds where SAT has polynomial size circuits and $\text{PH} \not\subseteq \text{P}^{\text{NP}}$ [10,16]. Therefore, it appears that in order to prove learnability of Boolean circuits with NP-assistants, we need non-relativizing techniques.

Because of the space limitations, we only give sketches of the proofs in this extended abstract. Some of the proofs are omitted completely.

In complexity theory, the class SPP has been used successfully in studying computational problems with intermediate complexity. The results in this

paper help to further establish the role of the class SPP in understanding the complexity of learning problems.

2 Notations, Definitions, and Technical Preliminaries

We follow standard language-theoretic notations in this paper. For sets X_1 and X_2 , $X_1 \setminus X_2$ denotes their set difference. Let λ denote the empty string. For $y_1, y_2 \in \Sigma^*$, $y_1 \leq y_2$ means that y_1 is lexicographically smaller than or equal to y_2 . Let \mathbf{Z} denote the set of integers and \mathbf{N} the set of natural numbers.

Complexity theory. We use standard definitions and notations from complexity theory. In particular, oracle Turing machines play a central role in many of the notions in this paper. Please refer to standard text books for these definitions and notations. For a nondeterministic polynomial time (in short, NP) machine M , $acc_M : \Sigma^* \rightarrow \mathbf{N}$ denotes the function defined as: $acc_M(x)$ = the number of accepting paths of M on input x . $\#P$ denotes the class of functions $\{acc_M | M \text{ is an NP machine}\}$. $gap_M : \Sigma^* \rightarrow \mathbf{Z}$ denotes the function defined as: $gap_M(x)$ = (number of accepting paths of M on input x – the number of rejecting paths of M on input x). GapP denotes the class of functions $\{gap_M | M \text{ is an NP machine}\}$ [8]. GapP is the closure of $\#P$ under subtraction. We will make implicit use of the following closure properties of GapP: a) the product of a polynomial number of GapP functions is a GapP function, b) the sum of a polynomial number of GapP functions is a GapP function (see [8] for details).

Using the notion of gaps we can define the complexity class SPP as follows. A language L is in SPP if there is a function f in GapP such that; $f(x) = 1$ if $x \in L$ and $f(x) = 0$ if $x \notin L$. For the purpose of comparison, we give the definition of the class UP. A language L is in UP, if there is a function f in $\#P$ such that; $f(x) = 1$ if $x \in L$ and $f(x) = 0$ if $x \notin L$. Thus, SPP generalizes the class UP. The inclusions $P \subseteq UP \subseteq SPP$ are a direct consequence of the definitions.

Learning theory. We only deal with the learnability of finite concepts. A *representation class* \mathcal{P} is a tuple $\langle R, \mu, p \rangle$ where p is a polynomial, $R \subseteq \Sigma^*$ and μ is a collection $\mu = \{\mu_n\}_{n \in \mathbf{N}}$ such that $\mu_n : R \rightarrow 2^{\Sigma^{p(n)}}$ is a many-one mapping. Any element in the range of μ is called a *concept* of \mathcal{P} . The set of all concepts of \mathcal{P} is called the *concept class* represented by \mathcal{P} . When there is no confusion, we denote the concept class represented by \mathcal{P} also by \mathcal{P} . For any n , let \mathcal{P}_n denote the concepts of \mathcal{P} in $\Sigma^{p(n)}$. For any concept $c \in \mathcal{P}_n$ let $size(c)$ denote $\min\{|r| : \mu_n(r) = c\}$.

Although Angluin's query model is very closely related to our model, we omit the definitions for this model since we do not use it in any essential manner. Instead, we directly give formal definitions related to the teaching assistant model. Please refer to Angluin's original paper for definitions related to this query model [1]. We start with the abstract definition of a teaching assistant.

Definition 1. Let \mathcal{P} be a representation class. Let $Q : \mathbf{N} \times \mathbf{N} \times \Sigma^* \times 2^{\Sigma^*} \rightarrow \{0, 1\}$ be a predicate. A tuple $\langle n, l, x, c \rangle \in \mathbf{N} \times \mathbf{N} \times \Sigma^* \times 2^{\Sigma^*}$, is said to be \mathcal{P} -valid if $c \in \mathcal{P}_n$ and $\text{size}(c) \leq l$. The teaching assistant defined by Q w. r. t. the representation class \mathcal{P} is the set $L(\mathcal{P}) = \{\langle n, l, x, c \rangle \mid \langle n, l, x, c \rangle \text{ is } \mathcal{P}\text{-valid and } Q(n, l, x, c)\}$.

Let $\mathcal{P} = \langle R, \mu, p \rangle$ be a representation class and $L(\mathcal{P})$ be a teaching assistant for \mathcal{P} . A learner is a deterministic oracle Turing machine transducer. The learning algorithm for \mathcal{P} comprises of a pair – a learner α , and a teaching assistant $L(\mathcal{P})$. The teacher selects a target concept $c \in \mathcal{P}_n$. The learner α , on input $\langle 0^n, 0^l \rangle$, has to output a representation $r \in R$ such that $\mu_n(r) = c$, provided that $\text{size}(c) \leq l$. α can query the teaching assistant $L(\mathcal{P})$. When α wants to query the teaching assistant about $\langle n, l, x, c \rangle$, it writes x on its oracle tape. Notice that other parameters, n, l and c , are fixed prior to learning and are implicitly communicated to the teaching assistant. For such a query x by the learner, the learner receives the answer ‘Yes’ if $\langle n, l, x, c \rangle \in L(\mathcal{P})$ and ‘No’ if $\langle n, l, x, c \rangle \notin L(\mathcal{P})$. After a finite number of steps, α outputs a representation of the target concept c . \mathcal{P} is said to be learnable with an assistant $L(\mathcal{P})$ if there is a learner α for \mathcal{P} with queries to $L(\mathcal{P})$ as teaching assistant. We say α learns \mathcal{P} with teaching assistant $L(\mathcal{P})$.

Let \mathcal{P} be a representation class. Let α be a learner learning \mathcal{P} with a teaching assistant $L(\mathcal{P})$. For an input $\langle 0^n, 0^l \rangle$, the *time complexity* of a learner is the number of steps it takes before it writes down a representation of the target concept. \mathcal{P} is said to be deterministic polynomial-time learnable with a teaching assistant $L(\mathcal{P})$, if there is a polynomial q and a deterministic learner α learning \mathcal{P} with $L(\mathcal{P})$ such that for all n , and for all $c \in \mathcal{P}_n$, on input $\langle 0^n, 0^l \rangle$, the running time of learner α learning c , is bounded by $q(n + l)$. Adapting the complexity-theoretic notation FP for functional polynomial time, we denote polynomial-time learnability by FP-learnability.

In order to quantify the complexity of teaching assistants we can define classes of teaching assistants with respect to a given representation class. These classes can be defined analogous to some well-studied complexity classes. In this paper we are interested in the teaching assistant class analogous to the complexity classes SPP and NP. We define the teaching assistant class SPP. The class NP can be defined analogously.

Definition 2. Let \mathcal{P} be a representation class and $L(\mathcal{P})$ be a teaching assistant. $L(\mathcal{P})$ is said to be in the class SPP(\mathcal{P}) if there exists a polynomial-time non-deterministic oracle Turing machine M which for any \mathcal{P} -valid tuple $\langle n, l, x, c \rangle$, on input $\langle 0^n, 0^l, x \rangle$, uses c as oracle and produces a $\text{gap}=1$ if $\langle n, l, x, c \rangle \in L(\mathcal{P})$ and $\text{gap}=0$ if $\langle n, l, x, c \rangle \notin L(\mathcal{P})$. The behavior of machine M is not specified on inputs that are not \mathcal{P} -valid.

We have the following definition of learnability with teaching assistant classes.

Definition 3. Let \mathcal{C} denote a teaching assistant class. We say a representation class \mathcal{P} is FP-learnable with a \mathcal{C} -assistant if there exists a teaching assistant $L(\mathcal{P}) \in \mathcal{C}(\mathcal{P})$ and an FP-learner α , such that α learns \mathcal{P} with the teaching assistant $L(\mathcal{P})$.

The following theorem relates the TA model to Angluin's model.

Theorem 1 ([4]). *Let \mathcal{P} be a representation class. Then \mathcal{P} is FP-learnable with membership queries if and only if it is FP-learnable with a P-assistant. If \mathcal{P} is FP-learnable with membership and equivalence queries, then it is FP-learnable with an NP-assistant.*

3 Learning Using SPP Assistants

In this section, we give a learning algorithm for monotone DNF using a teaching assistant in the class SPP. We also note that general DNFs and Boolean circuits are not learnable using SPP unless $\text{NP} \subseteq \text{SPP}$. Note that, from Theorem 1 and the fact that monotone DNFs are learnable with membership and equivalence queries, they are also learnable with an NP-assistant.

Let mDNF denotes the concept class of monotone functions represented by monotone DNF formulae. Let CIRCUITS denote the representation class of boolean circuits. For ease of presentation, we define the size of a concept c in mDNF as the number of prime implicants of c . This is only polynomially different from the regular notion of size and does not affect our algorithm.

Theorem 2. *The representation class mDNF is FP-learnable with an SPP-assistant.*

Proof. (Sketch) We will first present a teaching assistant $L(\text{mDNF})$ and show that there exists an oracle machine M with the required gap behavior accepting $L(\text{mDNF})$. We then present a learner LEARNER-mDNF which uses $L(\text{mDNF})$ to output a monotone formula for the target concept.

The learning algorithm that we give is an adaptation of Angluin's algorithm for learning monotone DNFs using membership and equivalence queries. Our algorithm works as follows. Let c be the concept represented by a monotone formula ϕ . As in the case of Angluin's algorithm for mDNF, the learning algorithm will find out each of the prime implicants of ϕ one by one. This is done by doing a prefix-search with the aid of the teaching assistant. In order to make the TA in SPP, the prime implicants are prefix-searched in the order of increasing number of variables in them. The teaching assistant that we use for the learner is defined as:

$L(\text{mDNF}) = \{ \langle n, l, \phi, k, y, c \rangle \mid \phi \text{ is a disjunction of monotone terms; there are no } x \in (c \setminus \phi) \text{ having less than } k \text{ 1s; there exists } z \in (c \setminus \phi) \text{ having exactly } k \text{ 1s; } y \text{ is a prefix of lex-first such } z \}$

Claim. $L(\text{mDNF}) \in \text{SPP}(\text{mDNF})$

Proof (of claim). The nondeterministic oracle machine M that witnesses $L(\text{mDNF})$ in $\text{SPP}(\text{mDNF})$ is defined below. M uses an auxiliary oracle nondeterministic machine N_2 as subroutine, which in turn uses N_1 . We define these two auxiliary machines first.

MACHINE $N_1^c(\langle n, l, \phi, i, k, y \rangle)$

1. **Guess** in lex-first order strings z_1, z_2, \dots, z_i
2. **Accept** if all the following conditions are met
3. y is a prefix of z_1
4. z_1, z_2, \dots, z_i have exactly k 1s
5. z_1, z_2, \dots, z_i are in c (using membership queries to c)
6. z_1, z_2, \dots, z_i are not in ϕ
7. **Else Reject**

MACHINE $N_2^c(\langle n, l, \phi, k, y \rangle)$

1. Produce a gap of $\sum_{i=1}^l acc_{N_1^c}(\langle n, l, \phi, i, k, y \rangle) \prod_{j=i+1}^l (1 - acc_{N_1^c}(\langle n, l, \phi, j, k, y \rangle))$

MACHINE $M^c(\langle n, l, \phi, k, y \rangle)$

1. If ϕ is not a disjunction of monotone terms produce a gap=0
2. Else produce a gap of $gap_{N_2^c}(\langle n, l, \phi, k, y \rangle) \prod_{k'=1}^{k-1} (1 - gap_{N_2^c}(\langle n, l, \phi, k', y \rangle))$

Let c be a target concept fixed by the teacher with $size(c) \leq l$. Let k_0 be the number of variables in the smallest (in terms of the number of variables) prime implicant of c that is not in ϕ . Let i_0 be the number of such prime implicants with k_0 variables.

We will argue that M produces a gap=1 if $(\langle n, l, \phi, k, y, c \rangle) \in L(\text{mDNF})$ and produces a gap=0 if $(\langle n, l, \phi, k, y, c \rangle) \notin L(\text{mDNF})$. Since if ϕ is not a disjunction of monotone terms M produces a gap=0, we will assume otherwise for the rest of the proof. $\langle n, l, \phi, k, y, c \rangle \in L(\text{mDNF})$ if and only if $k = k_0$ and y is a prefix of lex-first string $z \in (c \setminus \phi)$ with exactly k ones. We need to show that M^c produces a gap=1 in this case and produces a gap=0 otherwise. For brevity of presentation, we only present the case when y is the prefix of the lex-first string $z \in (c \setminus \phi)$ with exactly k ones. The other case can be argued similarly.

We first observe certain acceptance behavior of N_2 which is stated as the next claim. The proof follows from the descriptions of N_1 and N_2 .

Claim (subclaim).

1. If $k < k_0$, then $gap_{N_2^c}(\langle n, l, \phi, k, y \rangle) = 0$.
2. If $k = k_0$, then $gap_{N_2^c}(\langle n, l, \phi, k, y \rangle) = 1$.

Proof (sketch for subclaim).

- 1 $k < k_0$. In this case N_1^c will have 0 accepting paths. Hence, whatever be the value of i , $gap_{N_1^c}(\langle n, l, \phi, i, k, y \rangle) = 0$. Hence, the total gap produced by $N_2^c = 0$.
- 2 $k = k_0$. Split the sum in line 1 of the description of N_2^c into three cases: $i < i_0$, $i = i_0$, and $i > i_0$. For $i > i_0$, N_1^c will have 0 accepting paths and hence that part of the sum contributes 0 to the overall gap. For $i = i_0$, N_1^c will have exactly 1 accepting path and for this case each of terms in the product contributes 1. Thus the contribution to the overall gap in this case is 1. For the case $i < i_0$, consider the term $\prod_{j=i+1}^l (1 - acc_{N_1^c}(\langle n, l, \phi, j, k, y \rangle))$. Since $i < i_0$, there is term in this product with $j = i_0$. For this value of $j =$

i_0 , $acc_{N_1^c}(\langle n, l, \phi, i_0, k, y \rangle) = 1$ and hence $(1 - acc_{N_1^c}(\langle n, l, \phi, i_0, k, y \rangle)) = 0$. Hence the contribution from this case to the overall gap = 0. Therefore, the total gap = 1. \square

Case 1: $k = k_0$. In this case we need to show that $gap_{M^c} = 1$. Consider the gap expression in the line 2 of the description of M^c . From the above claim, $gap_{N_2^c}(\langle n, l, \phi, j, k, y \rangle) = 1$ and for all $k' \leq k - 1$, $gap_{N_2^c}(\langle n, l, \phi, j, k', y \rangle) = 0$. Hence the overall gap produced is 1.

Case 2: $k < k_0$. We need to show that $gap_{M^c} = 0$. From the above claim, the term $gap_{N_2^c}(\langle n, l, \phi, j, k, y \rangle) = 0$ and hence the overall gap produced is 0.

Case 3: $k > k_0$. (This case is similar to the case $i < i_0$ in the above claim) In this case we need to show that $gap_{M^c} = 0$. Consider the product term $\prod_{k'=1}^{k-1} (1 - gap_{N_2^c}(\langle n, l, \phi, k', y \rangle))$ of the gap expression in the line 2 of the description of M^c . Since k' runs from 1 to $k - 1$, if $k > k_0$ there is a term $(1 - gap_{N_2^c}(\langle n, l, \phi, k_0, y \rangle))$ that appear in this product. Now from the subclaim, for this value of k' , $(1 - gap_{N_2^c}(\langle n, l, \phi, k_0, y \rangle)) = 0$ and the product term, and consequently the overall gap, is 0.

The following algorithm learns mDNF in polynomial time by making only $O(ln)$ queries to the teaching assistant $L(\text{mDNF})$. In the algorithm, for a binary string $y \in \{0, 1\}^n$, Y denotes the the corresponding monotone term.

LEARNER-mDNF($0^n, 0^l$)

1. $\phi \leftarrow \mathbf{F}$
2. $k \leftarrow 1$
3. **while** $\langle 0^n, \phi, k, \lambda, c \rangle \notin L$
4. $k \leftarrow k + 1$
5. **if** $k = n + 1$
6. **then** output ϕ and stop
7. $y \leftarrow \lambda$
8. **for** $j = 1$ to n **do**
9. **if** $\langle 0^n, \phi, k, y1, c \rangle \in L$
10. **then** $y \leftarrow y1$
11. **else** $y \leftarrow y0$
12. $\phi \leftarrow \phi \vee Y$
13. **goto** 2

Finally we note that any representation class for which the equivalence problem (checking whether two strings represent the same concept) is co-NP-complete cannot be learned using an SPP assistant unless $\text{NP} \subseteq \text{SPP}$. We state this as our next theorem for the case of CIRCUITS and general DNFs. The proof essentially follows the line of argument for the proof of a similar result ([4], Theorem 7.4).

Theorem 3. *If the representation classes of CIRCUITS or DNF are learnable with an SPP-assistant then $\text{NP} \subseteq \text{SPP}$.*

4 Learning with NP Assistants

In this section we consider learnability using NP-assistants. Since learning with NP-assistants generalizes learning in Angluin's model with equivalence + membership queries, it is one of the most interesting teaching assistant classes.

Theorem 4. *If CIRCUITS are FP-learnable with an NP-assistant then $\text{SAT} \in \text{P/poly} \Rightarrow \text{PH} \subseteq \text{P}^{\text{NP}}$*

Proof. (Sketch) We will present the proof sketch here. We will use the self-reducibility structure of SAT to prove the theorem. Under this assumption we will show how to construct a circuit for SAT using P^{NP} machine. The construction is done inductively. For every n let C_n be the circuit that accepts SAT with n variables.

For the sake of simplicity, we omit the size parameter l in this proof. Let α be a learner, which on input 0^n learns a polynomial size circuit for SAT with n variables with an NP-assistant $L(\text{CIRCUITS}) = \{\langle n, y, c \rangle \mid y \in \Sigma^*; Q(n, y, c)\}$, where Q is the predicate defining the NP-assistant. Let the oracle machine N accept $L(\text{CIRCUITS})$. We will focus on α 's computation when the target circuit is the circuit for SAT.

Consider the following machine N' (now without oracles) which on input $\langle 0^n, y, C' \rangle$, simulates N with target concept SAT. Whenever N makes an oracle query f to SAT, it constructs two formulae f_0 and f_1 where f_i for $i \in \{0, 1\}$ is the formula f with the n^{th} variable replaced with i . Then, it checks whether $C(f_0) = 1$ or $C(f_1) = 1$. If yes then it continues simulation as if the query f is answered "YES"; otherwise it proceeds as if the query f is answered "NO". Let L be the NP language accepted by this machine N' .

Now consider the following FP machine M that constructs the circuit C_n for SAT inductively using the NP language L . When n is a constant, M exhaustively searches for the circuit for SAT. Now assume that M has constructed C_{n-1} . M simulates the learner α with SAT as target concept. Whenever α makes a query y to the teaching assistant, M makes a query $\langle 0^n, y, C_{n-1} \rangle$ to L and continues simulation. It is easy to see that M constructs the circuit for SAT. Therefore if CIRCUITS is learnable using an NP-assistant, then if $\text{SAT} \in \text{P/poly}$, SAT has P^{NP} -uniform circuits. Now, using standard techniques, it can be shown that $\Sigma_2^{\text{P}} \subseteq \text{P}^{\text{NP}}$, which implies $\text{PH} \subseteq \text{P}^{\text{NP}}$.

It is known that for any class \mathcal{C} which contains NP, $\text{SAT} \in \text{P/poly} \Rightarrow \text{PH} \subseteq \mathcal{C}$, then \mathcal{C} has languages with polynomial circuit complexity for any fixed polynomial. We have the following corollary whose proof is omitted.

Corollary 1. *If CIRCUITS is learnable with an NP-assistant, then for any k , P^{NP} has languages that does not have circuits of size n^k .*

Perhaps the most important concept class that is worth investigating in the TA model is general DNF formulae. An open problem regarding DNF learning in Angluin's model is whether equivalence + membership queries are powerful enough for learning DNF formulae. A natural step towards progress in this

problem is to investigate it in a more general learning framework than Angluin's model. The TA model with NP teaching assistants provides exactly such a framework. Are general DNF formulae learnable using an NP assistant?

Acknowledgments

I thank Chris Bourke for suggestions which helped me improve the presentation. I thank John Hitchcock for discussions. I thank the anonymous referees for comments which improved the presentation of the paper.

References

1. D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.
2. V. Arvind and P. Kurur. Graph isomorphism is in SPP. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 743–750, 2002.
3. V. Arvind and N. V. Vinodchandran. The complexity of exactly learning algebraic concepts. In *Proceedings of the 7th International Workshop on Algorithmic Learning Theory*, volume 1160 of *Lecture Notes in Artificial Intelligence*, pages 100–112, 1996.
4. V. Arvind and N. V. Vinodchandran. Exact learning via teaching assistants. *Theoretical Computer Science*, 241:51–81, 2000. Special issue devoted to the 7th International Workshop on Algorithmic Learning Theory.
5. R. Beigel. Perceptrons, PP, and the polynomial hierarchy. *Computational Complexity*, 4(4):339–349, 1994.
6. N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52:421–433, 1996.
7. J.-Y. Cai. $S_2^P \subseteq ZPP^{NP}$. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 620–629, 2001.
8. S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48:116–148, 1994.
9. S. Gupta. Closure properties and witness reduction. *Journal of Computer and System Sciences*, 50(3):412–432, 1995.
10. H. Heller. On relativized exponential and probabilistic complexity classes. *Information and Control*, 71:231–243, 1986.
11. J. M. Hitchcock. The size of SPP. *Theoretical Computer Science*. To Appear.
12. A. Klivans and D. van Melkebeek. Graph nonisomorphism has subexponential size proofs unless the polynomial-time hierarchy collapses. *SIAM Journal on Computing*, 31:1501–1526, 2002.
13. J. Köbler, U. Schöning, and J. Torán. Graph isomorphism is low for PP. *Journal of Computational Complexity*, 2:301–330, 1992.
14. M. Ogiwara and L. Hemachandra. A complexity theory of feasible closure properties. *Journal of Computer and System Sciences*, pages 295–325, 1993.
15. N. V. Vinodchandran. Counting complexity of solvable group problems. *SIAM Journal on Computing*, 2004. To Appear.
16. C. B. Wilson. Relativized circuit complexity. *Journal of Computer and System Sciences*, 31(2):169–181, 1985.

On the Complexity of Samples for Learning

Joel Ratsaby

University College London
Gower Street, London WC1E 6BT, United Kingdom
J.Ratsaby@cs.ucl.ac.uk

Abstract. In machine-learning, maximizing the sample margin can reduce the learning generalization-error. Thus samples on which the target function has a large margin (γ) convey more information so we expect fewer such samples. In this paper, we estimate the complexity of a class of sets of large-margin samples for a general learning problem over a finite domain. We obtain an explicit dependence of this complexity on γ and the sample size.

1 Introduction

Over the course of the last decade or so the field of learning theory has benefited from a rich body of work and the inception of a few key mathematical quantities which concisely capture and describe the complexity of accurate learning. Among those that characterize the problem of learning pattern classification is the Vapnik-Chervonenkis (VC) dimension. Based on the VC-dimension, upper bounds on the worst-case deviation of the learner's error from the optimal error may be obtained. The bounds are a function of the sample size m and some complexity term which is dependent on the VC-dimension of the function class. They are independent of the actual training-sample and hence become useful, i.e., have a value between 0 and 1, only for very large sample sizes¹. Thus in this respect they are weak. More recently ([10, 2, 3, 9]) sample dependent bounds have been shown to be stronger. They hold when the learner finds a hypothesis which maximizes the margin on a training sample over a class of functions.

Being that large-margin samples may yield better hypotheses (having lower error estimates) then they convey more information about the target. Intuitively, we expect that having more information about the target should result in fewer possible hypotheses. Since each hypothesis is associated with a particular set of training samples based on which the learner may infer it then we expect there to be fewer possible samples with increasing amount of information, i.e., with an increase in the margin.

In order to try to formalize these ideas we take a combinatorial approach aiming to estimate the cardinality of the class of sets of large-margin samples

¹ This has recently gotten significant attention from the computational learning field, see for instance the papers at the NeuroCOLT (2002) Workshop on Generalization bounds less than 1/2.

(where each set is associated with an hypothesis). Its logarithm is a natural measure of the complexity of any such sample. Our main technical result (Theorem 3) provides an estimate of this quantity.

We start with some notations and definitions.

2 Basic Notations and Definitions

Let X be a domain. For $a \in \mathbb{R}$, define $\text{sgn}(a) = +1$ if $a \geq 0$ and -1 if $a < 0$. The following definitions can be found for instance in [1]. For a class \mathcal{A} of real-valued functions on X the *Pseudo-dimension*, denoted as $\text{dim}_p(\mathcal{A})$, is defined as the maximum cardinality m of a sample $S = \{x_{i_1}, \dots, x_{i_m}\} \subset X$ such that there exists a *translate vector* $r = [r_1, r_2, \dots, r_m] \in \mathbb{R}^m$ where for each vector $v \in \{-1, 1\}^m$ there is a function $a_v(\cdot) \in \mathcal{A}$ that satisfies $\text{sgn}(a_v(x_{i_j}) - r_j) = v_j$ for $1 \leq j \leq m$. The sample S is said to be *shattered* by \mathcal{A} . In the special case where \mathcal{A} consists of mappings from $X \rightarrow \{-1, 1\}$ and $r_j = 0$, $1 \leq j \leq m$, then $\text{dim}_p(\mathcal{A})$ is called the *Vapnik-Chervonenkis dimension* of \mathcal{A} and is denoted by $VC(\mathcal{A})$.

For any $\gamma > 0$ the γ -*dimension* of \mathcal{A} , denoted as $\text{fat}_\gamma(\mathcal{A})$, is defined as the maximum cardinality m of a sample $S = \{x_{i_1}, \dots, x_{i_m}\} \subset X$ such that there exists $r = [r_1, r_2, \dots, r_m] \in \mathbb{R}^m$ where for each vector $v \in \{-1, 1\}^m$ there is a function $a_v(\cdot) \in \mathcal{A}$ that satisfies $a_v(x_{i_j}) \geq r_j + \gamma$ if $v_j = 1$ and $a_v(x_{i_j}) \leq r_j - \gamma$ if $v_j = -1$, for $1 \leq j \leq m$. The sample S is said to be γ -*shattered* by \mathcal{A} .

For $B > 0$ let \mathcal{H} denote a class of real-valued functions h from X into $[0, B]$. Let $\mathbb{S} = X^m$ consist of all sets $S \subset X$ with cardinality $|S| = m$. For any $S = \{x_{i_1}, \dots, x_{i_m}\}$ define the *margin* of h on S to be

$$d_S(h) = \min_{\{x_{i_j}: 1 \leq j \leq m\}} |h(x_{i_j}) - B/2|.$$

Denote by $\mathbb{I}(A)$ the indicator function which takes the value 1 for any $x \in A$ and zero otherwise. For brevity, we sometimes write $\mathbb{I}(|h(x) - B/2| - c)$ instead of $\mathbb{I}(x \in \{z : |h(z) - B/2| - c \geq 0\})$. Henceforth X is assumed to be finite and well-ordered and our result will depend² on $|X|$. All logarithms are taken with respect to the base 2.

3 Overview of the Problem of Learning

The generalized Probably-Approximately-Correct (PAC) model of learning from examples [5] can be used to represent learning classification. Under this model, data used for training and testing is generated independently and identically

² In this respect our analysis differs from standard learning models which admit an infinite domain. However, our work here is part of an ongoing research on comparing the value of different kinds of information-inputs for learning and parallels the work of Kolmogorov on algorithmic complexity of information where quantities are expressed in terms of the length n of binary sequences that describe some finite domain X of size $|X| = 2^n$.

(i.i.d) according to an unknown but fixed probability distribution P which is defined over the input/output pairing $(x, y) \in X \times \{-1, 1\}$ where the two possible classification categories are labeled as -1 and 1 . In general, y need not be a function of x however for the purpose of our work here we assume $y = \text{sgn}(t(x))$ where $t \in \mathcal{H}$ is some unknown *target* function in a class \mathcal{H} of real-valued functions on X . To measure the misclassification error by any h in \mathcal{H} the natural choice is the empirical error which is based on a randomly drawn sample $S_m = \{(x_{i_j}, y_{i_j})\}_{j=1}^m$ according to P . It is defined as

$$L_m(h) = \frac{1}{m} \sum_{j=1}^m \mathbb{I}(\text{sgn}(h(x_{i_j})) \neq y_{i_j}).$$

The (true) misclassification error of h is $L(h) = P((x, y) : \text{sgn}(h(x)) \neq y)$ and is unknown to the learner as P is not known. According to this model, the process of learning amounts to finding a hypothesis $\hat{h} \in \mathcal{H}$ which minimizes the empirical error over \mathcal{H} , i.e.,

$$L_m(\hat{h}) = \min_{h \in \mathcal{H}} L_m(h).$$

The minimal true-misclassification error is clearly zero, i.e., $L(t) = 0$. But in general the error of learning, $L(\hat{h})$, may be greater than zero. One of the main aims of the research in this field is to understand which settings, i.e., different learning algorithms and hypothesis classes \mathcal{H} , yield lower error $L(\hat{h})$. Theoretical estimates of $L(\hat{h})$ exist for various scenarios ([1]), the simplest being the pure PAC setting. This is described in the following result which appears as Theorem 4.1 in [4].

Theorem 1. *Let \mathcal{H} be a class of functions from X to $\{-1, 1\}$ with $VC(\mathcal{H}) = d$. For any probability distribution P on $X \times \{-1, 1\}$, let $S = \{(x_{i_j}, y_{i_j})\}_{j=1}^m$ be drawn according to P . Based on S , suppose $\hat{h} \in \mathcal{H}$ satisfies $L_m(\hat{h}) = 0$. Then for any $\delta > 0$, with probability $1 - \delta$,*

$$L(\hat{h}) \leq \epsilon(m, d, \delta) = \frac{2}{m} \left(d \log \frac{2em}{d} + \log \frac{2}{\delta} \right)$$

provided $m \geq \max\{d, 2/\epsilon(m, d, \delta)\}$.

Consider now the scenario where the learner obtains a sample S of cardinality m on which it is known that the target t has a large margin, i.e., $d_S(t) \geq \gamma$. Support Vector Machines and other Kernel-based learning methods which use the principle of maximizing the margin ([10, 4]) can be represented in this way. In general, the γ -dimension $\text{fat}_\gamma(\mathcal{H})$ of a class \mathcal{H} decreases with an increase in γ so as the following result suggests, the error of the classifier is significantly reduced with a larger margin (this result appears as Corollary 4.14 in [4]).

Theorem 2. *Let \mathcal{H} consist of real-valued functions from X to $[0, 1]$ and fix a $\gamma > 0$. Consider doing classification by thresholding functions h at $1/2$, i.e., $\text{sgn}(h(x) - 1/2)$, and denote by $L_m(h)$, $L(h)$ the corresponding empirical and*

true misclassification errors, respectively. For any probability distribution P on $X \times \{-1, 1\}$, let $S = \{(x_{ij}, y_{ij})\}_{j=1}^m$ be drawn according to P . Based on S suppose that $\hat{h} \in \mathcal{H}$ satisfies $L_m(\hat{h}) = 0$ and $d_S(\hat{h}) > \gamma$. Then for any $\delta > 0$, with probability $1 - \delta$,

$$L(\hat{h}) \leq \epsilon(m, d, \delta) = \frac{2}{m} \left(\text{fat}_{\gamma/8}(\mathcal{H}) \log \frac{8em}{\gamma \text{fat}_{\gamma/8}(\mathcal{H})} \log \frac{32m}{\gamma^2} + \log \frac{4}{\gamma} \right)$$

provided $m \geq \max\{2/\epsilon(m, d, \delta), \text{fat}_{\gamma/8}(\mathcal{H})\}$.

This result is an example of a sample-dependent bound since in reality the value of γ is the margin achieved by \hat{h} which obviously depends on the data. It is apparent that large-margin samples are worth more for learning since the bound decreases with increasing γ .

Our interest now is to estimate the complexity of large-margin samples. In order to characterize this quantitatively, starting in the next section our approach is to consider all samples $S \subset X$ of size m on which there exist hypotheses $h \in \mathcal{H}$ that have a margin $\gamma > 0$. We estimate the number of distinct sets A_h generated by $h \in \mathcal{H}$ where for a fixed h , A_h consists of all samples $S \in \mathbb{S}$ such that $d_S(h) > \gamma$. Since for a fixed h , any $S \in A_h$ may be used to learn h to within the same error bound of Theorem 2 (with $1 - \delta$ confidence) then A_h essentially is a set of ‘equivalent’ samples of size m .

4 Hyperconcepts

The space \mathbb{S} consists of all samples $S \subset X$ of size m . On \mathbb{S} consider sets of the form

$$A_{\beta, h} = \{S \in \mathbb{S} : d_S(h) \geq \beta\}, \quad \beta > 0$$

where the dependence on m is omitted in this notation. We refer to indicator functions on \mathbb{S} which take the form

$$h'_{\beta, h}(S) = \mathbb{I}(A_{\beta, h})$$

as *hyperconcepts* and we sometimes write just h' .

For any fixed margin parameter $\gamma > 0$ define the *hyperclass*

$$\mathcal{H}'_{\gamma} = \{h'_{\gamma, h} : h \in \mathcal{H}\}. \quad (1)$$

In words, \mathcal{H}'_{γ} consists of all sets of samples $S \subset X$ of cardinality m on which the corresponding hypotheses h (there may be several hypotheses associated with the same set) have a margin of at least γ .

Considering γ , m and \mathcal{H} as given and fixed but allowing the possible training sample $S \in \mathbb{S}$ to vary then the quantity $\log |\mathcal{H}'_{\gamma}|$ represents the description length of any hyperconcept $h' \in \mathcal{H}'_{\gamma}$ and is thus a measure of the richness or complexity of the hyperclass \mathcal{H}'_{γ} . Alternatively stated, it is the complexity of indexing the set t' (corresponding to the target t) which consists of γ -good samples each of which

may be chosen to train and learn t and yield an error which is bounded as in Theorem 2³. Note that by definition of \mathcal{H}'_γ the $\log |\mathcal{H}'_\gamma|$ decreases with increasing γ which is also intuitive from a learning perspective since larger-margin samples produce better error-bounds hence are implicitly more informative about t so there are fewer possible hypotheses and hence fewer possible sets of samples that induce them via learning. Our interest in this paper is to obtain an explicit dependence of this complexity on γ and m .

The following main result of the paper gives an estimate of this dependence.

Theorem 3. *Let B be a finite positive real number. For any $1 \leq m \leq |X|$ and $\gamma > 0$, consider a class \mathcal{H} of real-valued functions from X to the bounded interval $[0, B]$ and let $1 \leq \text{fat}_{\gamma/2}(\mathcal{H}) \leq |X| - m$. Then*

$$\log |\mathcal{H}'_\gamma| \leq \text{fat}_{\gamma/16}(\mathcal{H}) \log \left(\frac{8eB(|X| - m)}{\gamma \text{fat}_{\gamma/16}(\mathcal{H})} \right) \log \left(\frac{16B^2(|X| - m)}{\gamma^2} \right) + 2.$$

The next section contains the technical work of the proof of this theorem.

5 Proof of Theorem 3

Viewing \mathcal{H}'_γ as a class of boolean concepts on \mathbb{S} then, in general, it may have a limited separation ability over any *hypersample* $\zeta_N = \{S^{(1)}, \dots, S^{(N)}\}$ with $S^{(j)} \in \mathbb{S}$, $1 \leq j \leq N$ and $1 \leq N \leq |\mathbb{S}|$. The growth function $\Pi_{\mathcal{H}'_\gamma}(N)$, introduced by [11], captures its separation ability. It is defined as:

$$\Pi_{\mathcal{H}'_\gamma}(N) \equiv \max_{\zeta_N \subset \mathbb{S}} \Pi_{\mathcal{H}'_\gamma}(\zeta_N) \equiv \max_{\zeta_N \subset \mathbb{S}} \left| \left\{ [h'(S^{(1)}), \dots, h'(S^{(N)})] : h' \in \mathcal{H}'_\gamma \right\} \right|. \quad (2)$$

Since \mathbb{S} is finite, then viewing \mathbb{S} as a set ζ_r with $r = |\mathbb{S}|$ we have the following simple bound: $|\mathcal{H}'_\gamma| \leq \Pi_{\mathcal{H}'_\gamma}(|\mathbb{S}|)$. Our approach is therefore to upper bound the growth function $\Pi_{\mathcal{H}'_\gamma}(N)$.

Let N be a positive integer and consider any hypersample $\zeta_N = \{S^{(1)}, \dots, S^{(N)}\} \subset \mathbb{S}$. Denote by $S_i^{(j)}$ the i^{th} point in the sample $S^{(j)}$ based on the ordering of the elements of $S^{(j)}$ (which is induced by the ordering of the elements in X). Then

$$\begin{aligned} & \Pi_{\mathcal{H}'_\gamma}(\zeta_N) \tag{3} \\ &= \left| \left\{ \left[\mathbb{I} \left(\min_{x \in S^{(1)}} |h(x) - B/2| - \gamma \right), \dots, \mathbb{I} \left(\min_{x \in S^{(N)}} |h(x) - B/2| - \gamma \right) \right] : h \in \mathcal{H} \right\} \right| \\ &= \left| \left\{ \left[\prod_{j=1}^m \mathbb{I} \left(|h(S_j^{(1)}) - B/2| - \gamma \right), \dots, \prod_{j=1}^m \mathbb{I} \left(|h(S_j^{(N)}) - B/2| - \gamma \right) \right] : h \in \mathcal{H} \right\} \right| \end{aligned}$$

³ In this respect the samples in t' are equivalent so we are interested in counting not them but the number of hyperconcepts.

where we used the fact that the only way the minimum of the functions exceeds γ is if all functions exceed it. Order the elements in each set of ζ_N by the underlying ordering on X . Then put the sets in lexical ordering starting with the first up to the m^{th} element, so for instance, if $N = 3$, $m = 4$ and

$$\zeta_3 = \{ \{x_2, x_8, x_9, x_{10}\}, \{x_2, x_5, x_8, x_9\}, \{x_3, x_8, x_{10}, x_{13}\} \}$$

then the ordered version is

$$\{ \{x_2, x_5, x_8, x_9\}, \{x_2, x_8, x_9, x_{10}\}, \{x_3, x_8, x_{10}, x_{13}\} \}.$$

Let \hat{N} be a positive integer and consider the mapping $G : \mathbb{S}^N \rightarrow \mathbb{S}^{\hat{N}}$ which takes a hypersample ζ_N and produces a hypersample $\zeta_{\hat{N}}$ consisting of non-overlapping sets. It is defined as follows:

Procedure G:

Given a hypersample ζ_N construct $\zeta_{\hat{N}}$ as follows: Let $\hat{S}^{(1)} = S^{(1)}$. For any $2 \leq i \leq N$, let

$$\hat{S}^{(i)} = S^{(i)} \setminus \bigcup_{k=1}^{i-1} \hat{S}^{(k)}.$$

Let \hat{N} be the number of non-empty sets $\hat{S}^{(i)}$.

Note that \hat{N} may be smaller than N since there may be a set in ζ_N which is contained in the union of other sets in ζ_N . Leaving the empty sets out we henceforth let $\zeta_{\hat{N}}$ denote the new hypersample. It is easy to verify by induction that the sets of $\zeta_{\hat{N}}$ are mutually exclusive and their union equals that of the original sets in ζ_N .

For any point $x \in X$ let

$$\theta_h^\gamma(x) \equiv \mathbb{I}(x \in \{z : |h(z) - B/2| - \gamma \geq 0\})$$

or simply $\theta_h(x)$. Let

$$e_{S^{(i)}}(h) = \prod_{j=1}^m \theta_h(S_j^{(i)}),$$

with

$$v_{\zeta_N}(h) \equiv [e_{S^{(1)}}(h), \dots, e_{S^{(N)}}(h)]$$

where for brevity we sometimes write $v(h)$. Let

$$V_{\mathcal{H}}(\zeta_N) = \{v_{\zeta_N}(h) : h \in \mathcal{H}\}$$

or simply $V(\zeta_N)$. Then (4) becomes $|V_{\mathcal{H}}(\zeta_N)|$. We have the following:

Claim 1 *For any $\zeta_N \subseteq \mathbb{S}$, $|V_{\mathcal{H}}(\zeta_N)| \leq |V_{\mathcal{H}}(G(\zeta_N))|$.*

Proof: We make repetitive use of the following: let $A, B \subset X$ be two non-empty sets and let $C = B \setminus A$. Then for any h , any $b \in \{0, 1\}$, if $[e_A(h), e_B(h)] = [b, 0]$,

then $[e_A(h), e_C(h)]$ may be either $[b, 0]$ or $[b, 1]$ since the elements in B which caused the product $e_B(h)$ to be zero may or may not also be in C . In the other case if $[e_A(h), e_B(h)] = [b, 1]$ then $[e_A(h), e_C(h)] = [b, 1]$. Hence

$$|\{[e_A(h), e_B(h)] : h \in \mathcal{H}\}| \leq |\{[e_A(h), e_C(h)] : h \in \mathcal{H}\}|.$$

The same argument holds also for multiple A_1, \dots, A_k, B and $C = B \setminus \bigcup_{i=1}^k A_i$. Let $\zeta_{\hat{N}} = G(\zeta_N)$. We now apply this to the following:

$$\begin{aligned} & |\{[e_{S^{(1)}}(h), e_{S^{(2)}}(h), e_{S^{(3)}}(h), \dots, e_{S^{(N)}}(h)] : h \in \mathcal{H}\}| \\ &= |\{[e_{\hat{S}^{(1)}}(h), e_{S^{(2)}}(h), e_{S^{(3)}}(h), \dots, e_{S^{(N)}}(h)] : h \in \mathcal{H}\}| \end{aligned} \quad (4)$$

$$\leq |\{[e_{\hat{S}^{(1)}}(h), e_{\hat{S}^{(2)}}(h), e_{S^{(3)}}(h), \dots, e_{S^{(N)}}(h)] : h \in \mathcal{H}\}| \quad (5)$$

$$\leq |\{[e_{\hat{S}^{(1)}}(h), e_{\hat{S}^{(2)}}(h), e_{\hat{S}^{(3)}}(h), e_{S^{(4)}}(h), \dots, e_{S^{(N)}}(h)] : h \in \mathcal{H}\}| \quad (6)$$

$$\leq \dots$$

$$\leq |\{[e_{\hat{S}^{(1)}}(h), e_{\hat{S}^{(2)}}(h), e_{\hat{S}^{(3)}}(h), e_{\hat{S}^{(4)}}(h), \dots, e_{\hat{S}^{(N)}}(h)] : h \in \mathcal{H}\}| \quad (7)$$

where (4) follows since using G we have $\hat{S}^{(1)} \equiv S^{(1)}$, (5) follows by applying the above with $A = \hat{S}^{(1)}$, $B = S^{(2)}$ and $C = \hat{S}^{(2)}$, (6) follows by letting $A_1 = \hat{S}^{(1)}$, $A_2 = \hat{S}^{(2)}$, $B = S^{(3)}$, and $C = \hat{S}^{(3)}$. Finally, removing those sets $\hat{S}^{(i)}$ which are possibly empty leaves \hat{N} -dimensional vectors consisting only of the non-empty sets so (7) becomes $|\{[e_{\hat{S}^{(1)}}(h), \dots, e_{\hat{S}^{(\hat{N})}}(h)] : h \in \mathcal{H}\}|$. \square

Hence (4) is bounded from above as

$$\Pi_{\mathcal{H}_\zeta}(\zeta_N) \leq |V_{\mathcal{H}}(G(\zeta_N))|. \quad (8)$$

Now, for a positive integer N^* , define a mapping $Q : \mathbb{S}^N \rightarrow \mathbb{S}^{N^*}$ as follows:

Procedure Q:

Given a hypersample ζ_N . Construct ζ_{N^*} as follows: Let $Y = X \setminus S^{(1)}$ and let the elements in Y be ordered according to their ordering on X (we will refer to them as y_1, y_2, \dots). Let $S^{*(1)} = S^{(1)}$. For $2 \leq i \leq |X| - m + 1$, let $S_m^{*(i)} = y_{i-1}$ and $S_j^{*(i)} = S_{j+1}^{*(i-1)}$, $1 \leq j \leq m - 1$.

Henceforth denote by $N^* \equiv |X| - m + 1$. We now have the following:

Claim 2 For any $1 \leq N \leq |\mathbb{S}|$, and any $\zeta_N \subseteq \mathbb{S}$,

$$|V_{\mathcal{H}}(G(\zeta_N))| \leq |V_{\mathcal{H}}(G(Q(\zeta_N)))|$$

Proof: Let $\zeta_{\tilde{N}} \equiv G(Q(\zeta_N))$ and as before $\zeta_{\hat{N}} \equiv G(\zeta_N)$. Note that by definition of Procedure Q, it follows that $\zeta_{\tilde{N}}$ is a hypersample having $\tilde{N} = N^*$ non-overlapping sets. They consist of a single set $\tilde{S}^{(1)}$ of cardinality m and sets $\tilde{S}^{(i)}$, $2 \leq i \leq \tilde{N}$, each of which contains a single element of X . Moreover, their union $\bigcup_{i=1}^{\tilde{N}} \tilde{S}^{(i)} = X$. This holds for any ζ_N and $1 \leq N \leq |\mathbb{S}|$.

Consider the sets $V_{\mathcal{H}}(\zeta_{\tilde{N}})$, $V_{\mathcal{H}}(\zeta_{\hat{N}})$ and denote them simply by \hat{V} and \tilde{V} . For any $\hat{v} \in \hat{V}$ consider the following subset of \mathcal{H} ,

$$B(\hat{v}) = \{h \in \mathcal{H} : \hat{v}(h) = \hat{v}\}.$$

We consider two types of $\hat{v} \in \hat{V}$: The first does *not* have the following property: there exist functions $h_\alpha, h_\beta \in B(\hat{v})$ with $\theta_{h_\alpha}^\gamma(x) \neq \theta_{h_\beta}^\gamma(x)$ for at least one point $x \in X$. In this case, we have for all $h \in B(\hat{v})$

$$e_{\tilde{S}^{(1)}}(h) = e_{\hat{S}^{(1)}}(h)$$

and for $2 \leq j \leq \hat{N}$ with k_j ranging over the indices of all sets $\tilde{S}^{(k_j)} \subseteq \hat{S}^{(j)}$ we have

$$e_{\tilde{S}^{(k_j)}}(h) = e_{\hat{S}^{(j)}}(h).$$

Hence it easily follows that

$$|V_{B(\hat{v})}(\zeta_{\hat{N}})| = |V_{B(\hat{v})}(\zeta_{\hat{N}})|.$$

Let the second type of \hat{v} be the complement, namely, there exist functions $h_\alpha, h_\beta \in B(\hat{v})$ with $\theta_{h_\alpha}^\gamma(x) \neq \theta_{h_\beta}^\gamma(x)$ for at least one point $x \in X$. If such x exists only in $\hat{S}^{(1)}$ then the same argument as above holds and we still have

$$|V_{B(\hat{v})}(\zeta_{\hat{N}})| = |V_{B(\hat{v})}(\zeta_{\hat{N}})|.$$

If however there is also such x in some $\hat{S}^{(j)}$, $2 \leq j \leq \hat{N}$, then there exists some k_j such that $\tilde{S}^{(k_j)} \subseteq \hat{S}^{(j)}$ and

$$e_{\tilde{S}^{(k_j)}}(h_\alpha) \neq e_{\tilde{S}^{(k_j)}}(h_\beta)$$

due to the sets $\tilde{S}^{(i)}$, $2 \leq i \leq \tilde{N}$, all being singletons. Hence for this second type of \hat{v} we have

$$|V_{B(\hat{v})}(\zeta_{\hat{N}})| \geq |V_{B(\hat{v})}(\zeta_{\hat{N}})|. \quad (9)$$

Combining the above, then (9) holds for any $\hat{v} \in \hat{V}$.

Now, consider any two distinct $\hat{v}_\alpha, \hat{v}_\beta \in \hat{V}$. Clearly, $B(\hat{v}_\alpha) \cap B(\hat{v}_\beta) = \emptyset$ since every h has only one unique $\hat{v}(h)$. Moreover, for any $h_a \in B(\hat{v}_\alpha)$ and $h_b \in B(\hat{v}_\beta)$ we have $\tilde{v}(h_a) \neq \tilde{v}(h_b)$ for the following reason: there must exist some set $\hat{S}^{(i)}$, on which h_a and h_b differ (since $\hat{v}_\alpha \neq \hat{v}_\beta$). If $i = 1$ then they must differ on $\tilde{S}^{(1)}$. Else if $2 \leq i \leq \hat{N}$, then they differ on some set $\tilde{S}^{(j)} \subseteq \hat{S}^{(i)}$, where $2 \leq j \leq \tilde{N}$. Hence no two distinct $\hat{v}_\alpha, \hat{v}_\beta$ map to the same \tilde{v} . We therefore have

$$\begin{aligned} |V_{\mathcal{H}}(\zeta_{\hat{N}})| &= \sum_{\hat{v} \in \hat{V}} |V_{B(\hat{v})}(\zeta_{\hat{N}})| \\ &\leq \sum_{\hat{v} \in \hat{V}} |V_{B(\hat{v})}(\zeta_{\hat{N}})| \\ &= |V_{\mathcal{H}}(\zeta_{\hat{N}})| \end{aligned} \quad (10)$$

where (10) follows from (9) which proves the claim. \square

Note that the dimensionality of the boolean vectors in the set $V_{\mathcal{H}}(G(Q(\zeta_N)))$ is N^* which again is $|X| - m + 1$, regardless of the cardinality N of ζ_N . This follows by definition of Procedure Q .

Let us denote by ζ_{N^*} any hypersample obtained by Procedure Q followed by Procedure G , namely,

$$\zeta_{N^*} \equiv \left\{ S^{*(1)}, S^{*(2)}, \dots, S^{*(N^*)} \right\}$$

with any set $S^{*(1)} \subset X$ of cardinality m and

$$S^{*(k)} = \{x_{i_k}\}, \text{ where } x_{i_k} \in X \setminus S^{*(1)}, \quad k = 2, \dots, N^*.$$

Hence we may now write

$$\max_{1 \leq N \leq |\mathbb{S}|} \max_{\zeta_N \subseteq \mathbb{S}} \Pi_{\mathcal{H}'_\gamma}(\zeta_N) \leq \max_{1 \leq N \leq |\mathbb{S}|} \max_{\zeta_N \subseteq \mathbb{S}} |V_{\mathcal{H}}(G(Q(\zeta_N)))| \quad (11)$$

$$\leq \max_{\zeta_{N^*} \subseteq \mathbb{S}} |V_{\mathcal{H}}(\zeta_{N^*})| \quad (12)$$

where (11) follows from (4), Claim 1 and Claim 2 and (12) follows by definition of ζ_{N^*} . Now,

$$\begin{aligned} |V_{\mathcal{H}}(\zeta_{N^*})| &= |\{[e_{S^{*(1)}}(h), \dots, e_{S^{*(N^*)}}(h)] : h \in \mathcal{H}\}| \\ &\leq 2 |\{[e_{S^{*(2)}}(h), \dots, e_{S^{*(N^*)}}(h)] : h \in \mathcal{H}\}| \end{aligned} \quad (13)$$

where (13) follows trivially since $e_{S^{*(1)}}(h)$ is binary. So from (12) we have

$$\begin{aligned} \max_{1 \leq N \leq |\mathbb{S}|} \max_{\zeta_N \subseteq \mathbb{S}} \Pi_{\mathcal{H}'_\gamma}(\zeta_N) &\leq 2 \max_{\zeta_{N^*}} |\{[e_{S^{*(2)}}(h), \dots, e_{S^{*(N^*)}}(h)] : h \in \mathcal{H}\}| \\ &\leq 2 \max_{x_1, \dots, x_{N^*-1}} |\{[\theta_h^\gamma(x_1), \dots, \theta_h^\gamma(x_{N^*-1})] : h \in \mathcal{H}\}| \end{aligned} \quad (14)$$

where x_1, \dots, x_{N^*-1} run over any $N^* - 1$ points in X . Fix any subset $X_{N^*-1} = \{x_1, \dots, x_{N^*-1}\} \subseteq X$. We henceforth denote

$$C_\gamma(X_{N^*-1}) \equiv |\{[\theta_h^\gamma(x_1), \dots, \theta_h^\gamma(x_{N^*-1})] : h \in \mathcal{H}\}|. \quad (15)$$

We proceed to bound $C_\gamma(X_{N^*-1})$.

For any real number u define a quantization function

$$Q_\gamma(u) = \gamma \left\lfloor \frac{u}{\gamma} \right\rfloor.$$

For a function $h \in \mathcal{H}$ the function $Q_\gamma(h(x))$ maps from X into the finite subset $Z_\gamma = \{0, \gamma, 2\gamma, \dots, \lfloor B/\gamma \rfloor \gamma\}$ of $[0, B]$. We denote by $Q_\gamma(\mathcal{H})$ the function class $\{Q_\gamma(h) : h \in \mathcal{H}\}$. Consider the restriction of $Q_\gamma(\mathcal{H})$ on X_{N^*-1} and denote it by $\mathcal{A}_{\mathcal{H}}$ with functions $\alpha_h \in \mathcal{A}_{\mathcal{H}}$ mapping X_{N^*-1} into Z_γ .

For any subset $Y \subset X$ let \mathbb{R}^Y denote the space of real-valued functions on Y . For any $f \in \mathbb{R}^Y$ let the $l_\infty(Y)$ -norm of f be defined as $\|f\|_Y = \max_{x \in Y} |f(x)|$. For any class F of functions on Y let $\mathcal{M}(\epsilon, F, l_\infty(Y))$ be the packing number, i.e., the size of the maximal ϵ -separated set in F with respect to the $l_\infty(Y)$ -norm. Let the uniform packing number be defined as

$$\mathcal{M}(\epsilon, F, n) = \max\{\mathcal{M}(\epsilon, F, l_\infty(Y)) : Y, |Y| = n\}.$$

Every pair of distinct elements $\alpha_1, \alpha_2 \in A_{\mathcal{H}}$ satisfy $\|\alpha_1 - \alpha_2\|_Y \geq \gamma$ since they must be different on at least one point $x \in Y$ and on x their values $\alpha_1(x)$ and $\alpha_2(x)$ are restricted to the set Z_{γ} . Hence $\mathcal{A}_{\mathcal{H}}$ is itself a maximal γ -separated set and therefore

$$|\mathcal{A}_{\mathcal{H}}| = \mathcal{M}(\gamma, \mathcal{A}_{\mathcal{H}}, l_{\infty}(X_{N^*-1})). \quad (16)$$

Note also that

$$C_{\gamma}(X_{N^*-1}) \leq |\mathcal{A}_{\mathcal{H}}| \quad (17)$$

since given any $h \in \mathcal{H}$, for $x \in X_{N^*-1}$ with $\theta_h^{\gamma}(x) = 1$ then $\alpha_h(x)$ in general can take one of several possible values in Z_{γ} . Hence it now suffices to bound $\mathcal{M}(\gamma, \mathcal{A}_{\mathcal{H}}, N^* - 1)$.

Denoting by $d_{\mathcal{A}} = \text{fat}_{\gamma}(\mathcal{A}_{\mathcal{H}})$ then for $n \geq d_{\mathcal{A}}$ we have from Theorems 12.1 and 12.8 in [1] that

$$\mathcal{M}(8\gamma, \mathcal{A}_{\mathcal{H}}, n) < 2 \left(\frac{nB^2}{4\gamma^2} \right)^{d_{\mathcal{A}} \log(eBn/(\gamma d_{\mathcal{A}}))}. \quad (18)$$

This result holds for a general class of real-valued functions⁴ (not necessarily with a discrete finite range as for $\mathcal{A}_{\mathcal{H}}$).

Now, if $\gamma < 2\epsilon$ then

$$\text{fat}_{\epsilon}(Q_{\gamma}(\mathcal{H})) \leq \text{fat}_{\epsilon-\gamma/2}(\mathcal{H}) \quad (19)$$

since suppose $Q_{\gamma}(\mathcal{H})$ ϵ -shatters a set $\{x_{i_1}, \dots, x_{i_k}\} \subseteq X$ with translate vector $r = [r_1, \dots, r_k] \in \mathbb{R}^k$. Then for all $v \in \{-1, 1\}^k$ there is a function $f_v \in \mathcal{H}$ with

$$\begin{aligned} Q_{\gamma}(f_v(x_{i_j})) - r_j &\geq \epsilon & \text{if } v_j = 1 \\ Q_{\gamma}(f_v(x_{i_j})) - r_j &\leq -\epsilon & \text{if } v_j = -1 \end{aligned}$$

$1 \leq j \leq k$. It follows that

$$\begin{aligned} f_v(x_{i_j}) - r_j &\geq \epsilon & \text{if } v_j = 1 \\ f_v(x_{i_j}) - r_j &< -\epsilon + \gamma & \text{if } v_j = -1 \end{aligned}$$

or equivalently

$$\begin{aligned} f_v(x_{i_j}) - r_j - \gamma/2 &\geq \epsilon - \gamma/2 & \text{if } v_j = 1 \\ f_v(x_{i_j}) - r_j - \gamma/2 &< -\epsilon + \gamma/2 & \text{if } v_j = -1, \end{aligned}$$

$1 \leq j \leq k$, so \mathcal{H} $(\epsilon - \gamma/2)$ -shatters $\{x_{i_1}, \dots, x_{i_k}\}$ with a translate vector $r = [r_1 + \gamma/2, \dots, r_k + \gamma/2]$.

Hence continuing from (19) and letting $\epsilon = \gamma$ then

$$d_{\mathcal{A}} = \text{fat}_{\gamma}(Q_{\gamma}(\mathcal{H})) \leq \text{fat}_{\gamma/2}(\mathcal{H}).$$

⁴ Related to this, for a finite VC-dimension class H of binary-valued functions with a suitable definition of a functional margin, [7] generalized Sauer's Lemma [6] and estimated the cardinality of $F_{\gamma} \subset H$, where F_{γ} consists of all $h \in H$ having a margin $d_S(h) \geq \gamma$ on a set $S \subseteq X$. In [8] this quantity is shown to have a sharp threshold behavior with respect to the margin parameter.

So (18) becomes

$$\mathcal{M}(8\gamma, \mathcal{A}_{\mathcal{H}}, n) < 2 \left(\frac{nB^2}{4\gamma^2} \right)^{\text{fat}_{\gamma/2}(\mathcal{H}) \log(eBn/(\gamma \text{fat}_{\gamma/2}(\mathcal{H})))}. \quad (20)$$

Letting $n = N^* - 1$, from (16), (17) and (20) it follows that if $N^* - 1 \geq \text{fat}_{\gamma/2}(\mathcal{H})$ then

$$C_{\gamma}(X_{N^*-1}) \leq 2 \left(\frac{16(N^* - 1)B^2}{\gamma^2} \right)^{\text{fat}_{\gamma/16}(\mathcal{H}) \log(8eB(N^* - 1)/(\gamma \text{fat}_{\gamma/16}(\mathcal{H})))}. \quad (21)$$

Together with (2), (14), (15) and recalling that $N^* - 1 = |X| - m$, we therefore have

$$\max_{1 \leq N \leq |\mathbb{S}|} \Pi_{\mathcal{H}'_{\gamma}}(N) \leq 4 \left(\frac{16(|X| - m)B^2}{\gamma^2} \right)^{\text{fat}_{\gamma/16}(\mathcal{H}) \log(8eB(|X| - m)/(\gamma \text{fat}_{\gamma/16}(\mathcal{H})))}.$$

Now $|\mathcal{H}'_{\gamma}|$ is clearly bounded from above by $\Pi_{\mathcal{H}'_{\gamma}}(|\mathbb{S}|)$ hence we have

$$\log |\mathcal{H}'_{\gamma}| \leq \text{fat}_{\gamma/16}(\mathcal{H}) \log \left(\frac{8eB(|X| - m)}{\gamma \text{fat}_{\gamma/16}(\mathcal{H})} \right) \log \left(\frac{16B^2(|X| - m)}{\gamma^2} \right) + 2.$$

This proves Theorem 3. \square

Note, the function $\text{fat}_{\gamma}()$ is non-increasing with γ hence since the first factor in the bound dominates the first log factor then for all $\gamma \geq c$, for some constant c , $\log |\mathcal{H}'_{\gamma}|$ is non-increasing with increasing γ and decreasing with increasing m .

6 Conclusions

Existing data-dependent learning-error estimates indicate that larger-margin samples can be worth more for learning and hence implicitly convey more information about the target. In this paper we posed the question of determining the complexity of such good samples. We introduced a new notion of a class of sets of samples which we denote as *hyperclass* and then obtained an estimate of its cardinality as a function of the margin parameter γ and sample size m . The log of this quantity is our measure of the complexity of such γ -good samples.

References

1. Anthony M., Bartlett P. L., (1999), *Neural Network Learning: Theoretical Foundations*, Cambridge University Press, UK.
2. Antos A., Kgl B., Linder T., Lugosi G., (2002), Data-dependent margin-based generalization bounds for classification, *Journal of Machine Learning Research*, Vol. 3, pp.73-98.
3. Bartlett P. L., Boucheron S. Lugosi G., (2002), Model selection and error estimation, *Machine Learning*, Vol. 48, pp.85-113.

4. Cristianini N. and Shawe-Taylor J., (2000), *An Introduction to Support Vector Machines and other Kernel-based learning methods*, Cambridge University Press, UK.
5. Haussler D., (1992), Decision theoretic generalizations of the PAC model for neural net and other learning applications, *Information and Computation*, Vol. 100(1), pp.78-150.
6. Sauer N., (1972), On the density of families of sets, *J. Combinatorial Theory (A)*, Vol. 13, pp. 145-147.
7. Ratsaby J., (2004), A constrained version of Sauer's Lemma, *Proc. of Third Colloquium on Mathematics and Computer Science Algorithms, Trees, Combinatorics and Probabilities (MathInfo 2004)*, Vienna, Austria, September 2004, Birkhäuser.
8. Ratsaby J., (2004), A Sharp Threshold Result for Finite-VC Classes of Large-Margin Functions, Department of Computer Science Technical Report RN/04/06, University College London.
9. Shawe-Taylor J., Bartlett P. L., Williamson R. C., Anthony M., (1998), Structural risk minimization over data-dependent hierarchies, *IEEE Trans. Inf. Theory*, Vol. 44, pp.1926-1940.
10. Vapnik V.N., (1998), *Statistical Learning Theory*, Wiley..
11. Vapnik V. N and Chervonenkis A. Ya. (1971), On the uniform convergence of relative frequencies of events to their probabilities. *Theoret. Probl. Appl.*, Vol. 16, p.264-280.

New Results on On-Demand Broadcasting with Deadline via Job Scheduling with Cancellation

Wun-Tat Chan^{1,*}, Tak-Wah Lam²,
Hing-Fung Ting^{2,**}, and Prudence W.H. Wong²

¹ Department of Computing, Hong Kong Polytechnic University, Hong Kong
`cswtchan@comp.polyu.edu.hk`

² Department of Computer Science, University of Hong Kong, Hong Kong
`{twlam,hfting,whwong}@cs.hku.hk`

Abstract. This paper studies the on-demand broadcasting problem with deadlines. We give the first general upper bound and improve existing lower bounds on the competitive ratio of the problem. The novelty of our work is the introduction of a new job scheduling problem that allows cancellation. We prove that the broadcasting problem can be reduced to this scheduling problem. This reduction frees us from the complication of the broadcasting model and allows us to work on a conceptually simpler model for upper bound results.

1 Introduction

This paper revisits the on-demand broadcasting problem with deadlines. Improved lower bound and upper bounds are presented. In particular, the upper bounds are based on a reduction of the broadcasting problem to a job scheduling problem that allows cancellation. This reduction allows us to work on a conceptually simpler model for improved upper bounds.

On-demand broadcasting: The input is a sequence of page requests for a broadcasting server. These requests arrive at arbitrary times. The pages requested have possibly different length. At any time, the server can broadcast only one page and preemption is allowed. A unique feature of this problem is that all pending requests for a page can be satisfied with one single broadcast. Previous work on the broadcasting problem mainly focused on minimizing the total or maximum flow time [1, 3–5, 10, 11]. Broadcasting with deadline has not been studied until recently [7, 8]. Here each request has a deadline, and no profit is gained if a page is delivered beyond the deadline. The objective is to maximize the total (weighted) page length of the requests served by their deadlines. Kim and Chwa [8] studied the kind of preemptive broadcasting in which a preempted page will be re-broadcasted from the beginning (intuitively, this model does not assume that all preempted clients to have buffered a partial page properly). This

* This research was supported in part by Hong Kong RGC Grant PolyU-5172/03E.

** This research was supported in part by Hong Kong RGC Grant HKU-7045/02E.

version of broadcasting problem will be referred to as **Broadcasting** in the rest of this paper. Kim and Chwa gave a lower bound of $\min\{\sqrt{\Delta}, r\}$ on the competitive ratio of any on-line algorithm for **Broadcasting**, where Δ is the ratio between the length of the longest and shortest page, and r is the maximum number of requests for a page arriving at a time. For the special case when $\Delta = 1$, i.e., all pages have the same length, they showed that a heuristic based on highest profit is $(3 + 2\sqrt{2})$ -competitive. Finding an upper bound for the general case was left as an open problem. More recently, Kalyanasundaram and Velauthapillai [7] considered a stronger preemption model in which the server can resume broadcasting of a preempted page at the point of preemption and they also showed some interesting lower bound and upper bound results.

Summary of results: In this paper we study broadcasting with deadline and focus on the preemption model used by Kim and Chwa (i.e., a preempted page will be rebroadcasted from the beginning). Our first result is two improved lower bounds on the competitive ratio. We show that even if $r = 1$, no online algorithm for **Broadcasting** is $\sqrt{\Delta}$, thus improving the previous lower bound of $\min\{\sqrt{\Delta}, r\}$ to simply $\sqrt{\Delta}$. For the case $\Delta = 1$, we give a new lower bound of 2.59. This paper also contributes new results on the upper bounds. In particular, a $(4\Delta + 3)$ -competitive algorithm is given. Note that this is the first performance guarantee for the general case. Further, when $\Delta = 1$, a more stringent analysis shows that the algorithm is 5-competitive, improving the previous upper bound of $3 + 2\sqrt{2}$. The following table summarizes all the above results.

	Upper Bound	Lower Bound
General Δ	$4\Delta + 3$	<ul style="list-style-type: none"> • $\sqrt{\Delta}$ • $\min\{\sqrt{\Delta}, r\}$ [8]
$\Delta = 1$	<ul style="list-style-type: none"> • 5 • $3 + 2\sqrt{2}$ [8] 	2.59

The improvement on the upper bounds of **Broadcasting** is based on a reduction to a new job scheduling problem that allows cancellation. This job scheduling problem is conceptually simpler, allowing us to obtain a better analysis of the highest-profit heuristic. This reduction result should be contrasted with the observation by Kalyanasundaram and Velathapillai that existing upper bound results on the traditional scheduling problem do not hold for the broadcasting problem [7].

Scheduling jobs with possible cancellation: Online deadline scheduling is a classical real-time problem [2, 6, 9]. In the traditional model, users can release jobs at arbitrary times for scheduling on a single processor. These jobs have different processing times, profits and deadlines, and the processor can process one of these jobs at a time. The objective is to maximize the total profit of the jobs completed by their deadlines. To obtain a relation with **Broadcasting**, we consider a variant of job scheduling in which every job has an indefinite deadline but a user can cancel the job while waiting for processing. Denote this variant as **JS-Cancellation**. In fact, **JS-Cancellation** is natural in real-life applications. For

example, schedulers for printing jobs usually allow a user to cancel a printing job. Note that following the setting of **Broadcasting**, preemption is allowed and a preempted job will be restarted (instead of resumed at the point of preemption).

We say that an online problem P is reducible to another online problem Q , denoted as $P \prec Q$, if the following is true: Given any online algorithm for Q , we can construct an online algorithm for P with the same competitive ratio. It is easy to prove that if $P \prec Q$, then any upper bounds (on the competitive ratio) for Q implies the same upper bounds for P . The main result in this paper is that **Broadcasting** \prec **JS-Cancellation**.

Motivated by the above reduction, we work on upper bound results for **JS-Cancellation**. We show a refined analysis of the highest-profit heuristic; this leads to a $(4\Delta' + 3)$ -competitive algorithm for **JS-Cancellation**, where Δ' is the ratio between the processing time of the longest and shortest job. Together with the relationship **Broadcasting** \prec **JS-Cancellation**, we give the first upper bound for broadcasting in the general setting. Furthermore, when $\Delta' = 1$, the competitive ratio can be improved to 5.

Organization of the paper: The rest of the paper is organized as follows. In Section 2, we give some definitions and notations. In Section 3, we present lower bound results for **Broadcasting**. In Section 4, we present upper bound results for **JS-Cancellation**. In Section 5, we present the reduction of **Broadcasting** to **JS-Cancellation**.

2 Definitions and Notations

In this section, we give the definitions and notations needed for our discussion. The input for **Broadcasting** is a sequence σ of page requests. Every page P has a length $\ell(P)$. Every page request R has an arrival time $a(R)$, a deadline $d(R)$, a weight $w(R)$, and a name of a page $p(R)$ that it asks for broadcasting. We say that R is *alive* at time t if (1) $a(R) \leq t \leq d(R)$ and (2) it has not received the complete $p(R)$ yet. We say that it is *healthy* at time t if it is alive at t and $t + \ell(p(R)) \leq d(R)$. Note that if R is healthy at t , then we can broadcast $p(R)$ at time t and R can still receive the complete $p(R)$ before its deadline. Given any schedule S for σ , its profit $\rho(S)$ is the total weighted page length of the requests served by their deadlines, i.e., $\rho(S) = \sum_{R \in \mathcal{R}} w(R)\ell(p(R))$ where \mathcal{R} is the set of requests served by their deadlines.

The input for **JS-Cancellation** is a sequence σ of jobs interleaved with cancel requests. Every job J has an arrival time $a(J)$, a processing time $\ell(J)$ and a profit $\rho(J)$. A cancel request C has an arrival time $a(C)$ and the name of a job $j(C)$ that it asks for cancellation. Given any schedule S for σ , its profit $\rho(S)$ is the total profit of jobs completed.

3 Lower Bound Results for Broadcasting

In this section, we give a lower bound on the competitive ratio of any online algorithm for **Broadcasting**. Recall that Δ is the ratio between the length of the

longest and shortest page, and r is the maximum number of requests for a page arriving at a time. Our results show that even if $r = 1$, no online algorithm is better than $\sqrt{\Delta}$ -competitive. Note that $r = 1$ means that no two requests for the same page arrive at the same time.

Theorem 1. *No online algorithm for Broadcasting is better than $\sqrt{\Delta}$ -competitive. This holds even if $r = 1$.*

Proof. For the sake of simplicity, we show a lower bound of $\lfloor \sqrt{\Delta} \rfloor$. By a refinement of the argument, we can prove a lower bound of $\sqrt{\Delta}$ and the details will be given in the full paper.

Let $\alpha = \lfloor \sqrt{\Delta} \rfloor$. Assume that there are two pages, P_0 and P_1 whose length are α^3 and α , respectively. Given any online algorithm \mathcal{A} , we construct a sequence of requests as follows. At time 0, a request R for P_0 arrives with deadline at time α^3 , i.e., it has a tight deadline. Requests for P_1 arrive in phases. There are at most α^2 phases. In general, for $0 \leq i \leq \alpha^2 - 1$, the i -th phase starts at time $i\alpha$ and contains α requests arriving at consecutive time units whose deadlines are all at time $(i + 2)\alpha - 1$. If \mathcal{A} broadcasts P_1 in this phase, no further requests arrive after this phase; otherwise, we enter the next phase. Note that no further requests arrive after the $(\alpha^2 - 1)$ -th phase.

Now we analyze the performance of \mathcal{A} against the optimal offline algorithm \mathcal{O} . There are two cases. (1) If \mathcal{A} broadcasts P_0 to serve R , then \mathcal{O} will satisfy all requests for P_1 by broadcasting P_1 at the end of each phase, i.e., at time $(i + 1)\alpha - 1$ for the i -th phase. There are altogether α^2 phases, each with α requests. Therefore, $\rho(\mathcal{O}) = \alpha^3 \cdot \alpha = \alpha^4$. Thus, $\rho(\mathcal{O})/\rho(\mathcal{A}) = \alpha^4/\alpha^3 = \alpha$.

(2) If \mathcal{A} broadcasts P_1 in phase i , only the α requests released in this phase can be satisfied, all other previous requests are either not alive or not healthy. Thus, $\rho(\mathcal{A}) \leq \alpha \cdot \alpha = \alpha^2$. In this case, \mathcal{O} broadcasts P_0 and satisfies the request R . Therefore, $\rho(\mathcal{O})/\rho(\mathcal{A}) \geq \alpha^3/\alpha^2 = \alpha$. As a result, no online algorithm is better than α -competitive.

For the special case with constant page length, i.e., $\Delta = 1$, we have a lower bound result that no online algorithm can achieve a competitive ratio better than 2.59.

Theorem 2. *No online algorithm for Broadcasting with $\Delta = 1$ has competitive ratio better than 2.59.*

Proof. Consider two sequences of requests, $X = (x_0, x_1, \dots)$ and $Y = (y_0, y_1, \dots)$, each request is on a distinct page of unit length. All requests are with tight deadlines. Arrival times of x_0, x_1, \dots are $0, 1, \dots$, respectively. Arrival times of y_0, y_1, \dots are $0.5, 1.5, \dots$, respectively. The weights of the requests follow a series Ψ of non-negative numbers ψ_0, ψ_1, \dots such that $\psi_0 = 1, \psi_1 = c, \psi_2 = c^2 - 1$, and $\psi_i = c(\psi_{i-1} - \psi_{i-3})$ for $i > 2$, where $c > 1$ is a constant. The weights of requests x_0, x_1, x_2, \dots are $\psi_0, \psi_2, \psi_4, \dots$, respectively, and the weights of requests y_0, y_1, y_2, \dots are $\psi_1, \psi_3, \psi_5, \dots$, respectively.

The adversary works as follows. If the online algorithm \mathcal{A} satisfies a request j , no further requests arrive. If j is x_k for some k , $\rho(\mathcal{A}) = \psi_{2k}$. In this case, the

optimal offline algorithm \mathcal{O} satisfies requests y_0, \dots, y_k , thus $\rho(\mathcal{O}) = \sum_{i=0}^k \psi_{2i+1}$. If j is y_k for some k , $\rho(\mathcal{A}) = \psi_{2k+1}$. In this case, \mathcal{O} satisfies requests x_0, \dots, x_{k+1} , thus $\rho(\mathcal{O}) = \sum_{i=0}^{k+1} \psi_{2i}$. It can be shown by mathematical induction that both $\sum_{i=0}^k \psi_{2i+1}/\psi_{2k}$ and $\sum_{i=0}^{k+1} \psi_{2i}/\psi_{2k+1}$ are c . Hence if \mathcal{A} satisfies a request j , then \mathcal{A} is c -competitive. We want to find the maximum value of c such that the series Ψ contains a non-positive element. That implies that there is a request j without profit. \mathcal{A} must not switch to that request and hence \mathcal{A} will satisfy a request prior to request j . It can be verified that if c is less than or equal to 2.59, Ψ contains a non-positive element.

4 Upper Bound Results for JS-Cancellation

In this section, we present an algorithm, called GREEDY, for JS-Cancellation and analyze its competitive ratio. Recall that Δ' is the ratio between the processing time of the longest and shortest job. We consider the general case in which Δ' is arbitrary and the special case in which $\Delta' = 1$. We first define the online algorithm GREEDY as follows.

When the first job arrives or when a job completes, GREEDY schedules a job with the highest profit (ties are broken arbitrarily). When a job is being processed and a new job arrives, GREEDY will preempt the current job and start the new job if the new job carries at least twice the profit of the current job.

We now analyze the performance of GREEDY. Let S_G and S_O be the set of jobs completed by GREEDY and the optimal offline algorithm OPT, respectively. For any set of jobs S , let $\rho(S)$ be the sum of the profit of all the jobs in S . We prove the competitive ratio by constructing a mapping f to map each job in S_O to some job in S_G such that for every job $J \in S_G$, the total profit of all jobs in S_O that map to J is at most $(4\Delta' + 3)$ times the profit of J . More precisely, let $F(J)$ denote the set of jobs J' such that $f(J') = J$. We will show that for any job $J \in S_G$, $\rho(F(J)) \leq \rho(J) \cdot (4\Delta' + 3)$. As a result, $\rho(S_O) = \sum_{J \in S_G} \rho(F(J)) \leq \sum_{J \in S_G} \rho(J) \cdot (4\Delta' + 3) = (4\Delta' + 3) \cdot \rho(S_G)$. The following theorem states the competitive ratio of GREEDY for JS-Cancellation.

Theorem 3. *GREEDY is $(4\Delta' + 3)$ -competitive for JS-Cancellation.*

Proof. We first define the mapping as follows. For each job $J \in S_G$, define a sequence of jobs $Q_J = (J_1, J_2, \dots, J_k)$ with $J_k = J$ such that for $1 \leq i \leq k-1$, J_i is preempted by J_{i+1} during the execution of GREEDY. (Note that J_1, J_2, \dots, J_{k-1} may not be in S_G .) For all $1 \leq i \leq k$, we let $\beta(J_i) = J$. Notice that for $1 \leq i \leq k-1$, $2\rho(J_i) \leq \rho(J_{i+1})$, hence, $\rho(Q_J) \leq \sum_{1 \leq i \leq k} \rho(J_i)/2^{k-i} \leq 2\rho(J)$. The mapping f is constructed as follows. For any job $J \in S_O \cap S_G$, we set $f(J) = J$. For any job $J \in S_O - S_G$, suppose when J starts being processed by OPT, GREEDY is processing a job J' , then we set $f(J) = \beta(J')$. Note that J' must exist, otherwise GREEDY will schedule to process J .

Consider any job $J \in S_G$. We claim that for every job $J_i \in Q_J$, during the time interval I_i that GREEDY is processing J_i , the profit obtained by OPT is at most $2\Delta'\rho(J_i)$. Note that in the time interval I_i , OPT may be processing some jobs whose start time or finish time is outside I_i but we only count the profit obtained in I_i .

Let τ be the processing time of J_i . Suppose that in the time interval I_i , OPT has processed jobs s_1, \dots, s_m , with processing time τ_1, \dots, τ_m and profit ρ_1, \dots, ρ_m , respectively. We have $(\tau_1 + \dots + \tau_m) = \tau$. By the definition of Δ' , we have $\min(\tau_1, \dots, \tau_m) \geq \tau/\Delta'$. Combining these two constraints, we have $\Delta' \geq m$. In the execution of GREEDY in the interval I_i , none of the jobs s_1, \dots, s_m preempts the job J_i . Therefore, by the definition of GREEDY, $\rho_x \leq 2\rho(J_i)$ for all $1 \leq x \leq m$. In other words, $\max(\rho_1, \dots, \rho_m) \leq 2\rho(J_i)$. Hence, $\rho_1 + \dots + \rho_m \leq 2m\rho(J_i) \leq 2\Delta'\rho(J_i)$. Recall that $\rho(Q_J) \leq 2\rho(J)$. The sum of the profit of $F(J)$ within the interval $\bigcup_{1 \leq i \leq k} I_i$ is at most $\sum_{1 \leq i \leq m} 2\Delta'\rho(J_i) \leq 4\Delta'\rho(J)$. Notice that OPT may be still processing an un-completed job when GREEDY completes J , and this job has a profit at most $2\rho(J)$. Furthermore, $F(J)$ may contain J and OPT may process J outside the interval $\bigcup_{1 \leq i \leq k} I_i$. As a result, $\rho(F(J)) \leq 4\Delta'\rho(J) + 2\rho(J) + \rho(J) = \rho(J) \cdot (4\Delta' + 3)$. Hence, the theorem follows.

Now, we consider the special case $\Delta' = 1$. We can see that GREEDY is 7-competitive by substituting $\Delta' = 1$ to Theorem 3. Yet, by a more stringent analysis of GREEDY for $\Delta' = 1$, GREEDY is shown to be 5-competitive. Notice that $\Delta' = 1$ means that all jobs require the same processing time.

Theorem 4. *GREEDY is 5-competitive for JS-Cancellation when $\Delta' = 1$.*

Proof. The proof uses a similar framework as that of Theorem 3. For any job in S_O , we define the same mapping f as before. Consider any job $J \in S_G$. Define the sequence $Q_J = (J_1, J_2, \dots, J_k)$ as before. Because the processing time is the same for every job, there are at most k jobs other than J , say J'_1, J'_2, \dots, J'_k , in S_O mapping to J such that when J'_p starts processed by OPT, GREEDY is processing J_p , for $1 \leq p \leq k$. Since J'_p does not preempt J_p , we have $\rho(J'_p) \leq 2\rho(J_p)$. Furthermore, $F(J)$ may contain the job J itself. Therefore, $\rho(F(J)) \leq \rho(J) + \sum_{1 \leq p \leq k} \rho(J'_p) \leq \rho(J) + \sum_{1 \leq p \leq k} 2\rho(J_p) = \rho(J) + 2\rho(Q_J) \leq 5\rho(J)$. Therefore, the theorem follows.

5 The Reduction

In this section, we describe a reduction from Broadcasting to JS-Cancellation. Precisely, suppose that we have an online algorithm \mathcal{A} for JS-Cancellation with competitive ratio c . We show how to construct from \mathcal{A} an online algorithm \mathcal{B} for Broadcasting with competitive ratio at most c .

Roughly speaking, \mathcal{B} works as follows. From the input sequence σ_B of page requests for Broadcasting, \mathcal{B} creates a sequence σ_J of jobs and cancel requests for JS-Cancellation. Whenever \mathcal{B} creates any job or cancel request, it passes it to \mathcal{A} immediately for scheduling. \mathcal{B} monitors the behavior of \mathcal{A} . If \mathcal{A} takes any action

such as starting a job, \mathcal{B} takes some corresponding action such as broadcasting a page. When creating the jobs or cancel requests, \mathcal{B} makes sure that at any time t and for every page P , if there are healthy requests R with $p(R) = P$, then there will be a single job J for **JS-Cancellation** such that $\ell(J) = \ell(P)$ and $\rho(J) = \sum_{R \in \mathcal{H}} w(R) \cdot \ell(P)$, where \mathcal{H} is the set of healthy requests. Thus, if both \mathcal{A} starts job J and \mathcal{B} broadcasts P at some time t and there is no preemption, then both \mathcal{A} and \mathcal{B} receive the same profit and the two tasks finish at the same time, namely $t + \ell(P)$.

To get an idea on how the competitive ratio is preserved, let S_J and S_B be the schedules decided by \mathcal{A} and \mathcal{B} , and let O_J and O_B be the optimal offline schedules for σ_J and σ_B , respectively. We will prove below that

1. $\rho(S_J) = \rho(S_B)$ (i.e., the profit of the two schedules are equal) and
2. $\rho(O_B) \leq \rho(O_J)$.

Recall that \mathcal{A} has competitive ratio c . Then, we have $\frac{\rho(O_B)}{\rho(S_B)} \leq \frac{\rho(O_J)}{\rho(S_J)} \leq c$, and hence the competitive ratio of \mathcal{B} is no more than c .

Before giving the details, we need some definitions. For any page P , let $H(P, t)$ be the set of page requests R that are healthy at time t and $p(R) = P$. When \mathcal{B} creates a job J at t , its profit and length will be set according to some healthy page P at t . To remember this relationship, we define $g(J) = P$.

Now, we summarize the actions of \mathcal{B} . At any time t , for any page P , the following events may occur.

- A page request R with $p(R) = P$ arrives.
- A healthy page request R with $p(R) = P$ becomes not healthy.
- A broadcast of the page P has been preempted.
- The page P has been broadcasted completely.

In all these cases, \mathcal{B} first creates cancel requests to delete all waiting jobs (i.e., not being processed) associated with P . More precisely, for each waiting job J with $g(J) = P$, \mathcal{B} creates a cancel request C with $j(C) = J$. Then, if $H(P, t)$ is not empty, \mathcal{B} creates a job J_o with $g(J_o) = P$, $\ell(J_o) = \ell(P)$ and its profit $\rho(J_o) = \sum_{R \in H(P, t)} w(R) \cdot \ell(P)$.

Note that all the jobs and cancel requests created will have arrival time t . Recall that \mathcal{B} will also monitor \mathcal{A} to see how \mathcal{A} schedules the jobs. When \mathcal{A} starts or restarts a job J , then \mathcal{B} broadcasts the page $g(J)$. When \mathcal{A} preempts a job J , then \mathcal{B} preempts the current broadcasting, which must be for page $g(J)$. Below, we analyze the profits of the schedules.

Lemma 1. $\rho(S_B) = \rho(S_J)$.

Proof. By how \mathcal{B} creates jobs and cancel requests, it is clear that at any time t , every waiting job J has the property that

$$\rho(J) = \sum_{R \in H(g(J), t)} w(R) \cdot \ell(P).$$

The profit of scheduling J at time t is equal to the profit of broadcasting page $g(J)$ at time t . For every job J , \mathcal{A} schedules J at time t if and only if \mathcal{B} broadcasts $g(J)$ at time t . Thus, we have $\rho(S_J) = \rho(S_B)$.

Lemma 2. $\rho(O_B) \leq \rho(O_J)$.

Proof. We prove the lemma by constructing a job schedule S for σ_J such that $\rho(O_B) \leq \rho(S)$ and $\rho(S) \leq \rho(O_J)$. Then we can conclude that $\rho(O_B) \leq \rho(O_J)$. Notice that O_J is the optimal schedule for σ_J , thus, we have $\rho(O_J) \geq \rho(S)$. The schedule S is constructed to satisfy a property, which leads to the proof of $\rho(O_B) \leq \rho(S)$. Consider the set of requests \mathcal{R}_B and \mathcal{R}_O served by S_B and O_B , respectively. S will have the property that $\rho(S) = \sum_{R \in (\mathcal{R}_O \cup \mathcal{R}_B)} w(R)\ell(p(R))$. Then, we have $\rho(S) \geq \sum_{R \in \mathcal{R}_O} w(R)\ell(p(R)) = \rho(O_B)$.

Now, we show how S schedules jobs in σ_J and prove that it satisfies the property mentioned above. Note that the requests that are served by S_B correspond to a set of jobs in σ_J that are scheduled by S_J . In other words, there are no cancel requests in σ_J that cancel these jobs; S must be able to schedule these jobs and obtain at least a profit of $\sum_{R \in \mathcal{R}_B} w(R)\ell(p(R))$.

We then consider the requests that have been served by O_B but not by S_B . Suppose that k pages, P_1, P_2, \dots, P_k are broadcasted by O_B such that P_i is broadcasted at time t_i for $1 \leq i \leq k$. Notice that all requests in $\mathcal{R}_O - \mathcal{R}_B$ that are served by O_B at time t_i are healthy, thus, there is a job $J_i \in \sigma_J$ at time t_i such that $g(J_i) = P_i$. We make S to schedule J_i at time t_i , thus, obtaining a profit of the corresponding requests. Therefore, S can obtain a total profit of $\sum_{R \in (\mathcal{R}_O - \mathcal{R}_B)} w(R)\ell(p(R))$. Combining with the profit obtained from jobs that are scheduled by S_J , the total profit of S equals to $\sum_{R \in (\mathcal{R}_O \cup \mathcal{R}_B)} w(R)\ell(p(R))$. Hence, the lemma follows.

By the above two lemmas, we have the following theorem.

Theorem 5. *Given a c -competitive online algorithm for JS-Cancellation, we can design an online algorithm that is c -competitive for Broadcasting.*

References

1. Y. Bartal and S. Muthukrishnan. Minimizing maximum response time in scheduling broadcasts. In *Proceedings of the Annual ACM/SIAM Symposium on Discrete Algorithms*, pages 558–559, 2000.
2. S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4:125–144, 1992.
3. J. Edmonds and K. Pruhs. Multicast pull scheduling: when fairness is fine. *Algorithmica*, 36:315–330, 2003. A preliminary version appears in SODA 2002.
4. J. Edmonds and K. Pruhs. A maiden analysis of longest wait first. In *Proceedings of the 15th Annual Symposium on Discrete Algorithms*, pages 811–820, 2004.
5. B. Kalyanasundaram, K. Pruhs, and M. Velauthapillai. Scheduling broadcasts in wireless networks. *Journal of Scheduling*, 4(6):339–354, 2001. A preliminary version appears in ESA 2000.

6. B. Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):214–221, 2000.
7. B. Kalyanasundaram and M. Velauthapillai. On-demand broadcasting under deadline. In *Proceedings of the 11th Annual European Symposium on Algorithms*, pages 313–324, 2003.
8. J.H. Kim and K.Y. Chwa. Scheduling broadcasts with deadlines. In *Proceedings of the 9th Annual International Conference on Computing and Combinatorics*, pages 415–424, 2003.
9. G. Koren and D. Shasha. D^{over} : An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM Journal on Computing*, 24(2):318–339, 1995.
10. K. Pruhs and P. Uthaisombut. A comparison of multicast pull models. In *Proceedings of the 10th Annual European Symposium on Algorithms*, pages 808–819, 2002.
11. S. Muthukrishnan Swarup Acharya. Scheduling on-demand broadcasts: New metrics and algorithms. In *The Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 43–54, 1998.

Maximization of the Size and the Weight of Schedules of Degradable Intervals

Fabien Baille, Evripidis Bampis, and Christian Laforest

LaMI, CNRS UMR 8042, Université d'Evry,
Tour Evry 2, 523, Place des Terrasses 91000 Evry, France
{fabien.baille,bampis,laforest}@lami.univ-evry.fr

Abstract. We consider the problem of scheduling a set of intervals with controllable processing times in the presence of release dates and deadlines on a set of identical parallel machines in order to maximize either the number (the *size* problem) or the total weight (the *weight* problem) of accepted intervals. We call these jobs *degradable intervals*. We study a special case, called the *immediate case*, in which each accepted interval has to start at its release date. We call the general case the *non-immediate case*. For both criteria, we present optimal algorithms for the immediate case. For the non-immediate case, we prove the NP-hardness of the size problem and present a 1.58-approximation algorithm. For the weight problem, we propose a 2.54-approximation algorithm.

1 Introduction

In this paper, we are interested by the network access reservation problem. Assume clients wishing to reserve for a time the access to one of the k channels of a link managed by a provider. Due to congestion, the provider may not satisfy all clients. In such situations, he can degrade the service or reject some users. His goal is either the maximization of the accepted (even with degradation) connections or the weight (or profit) of the accepted connections. This situation can be seen as a scheduling problem that we describe now.

The model. The problem that we consider in this paper consists in scheduling a set σ of *degradable intervals*, on k *identical machines*. Each interval σ_i is represented by a release date $r_i \geq 0$, a maximal processing time $p_i > 0$, a deadline $d_i = r_i + p_i$, a minimal processing time $0 < q_i \leq p_i$ and a weight $w_i > 0$. We consider two variants: The *immediate model* where an accepted interval has to start at his release date and the *non-immediate model* in which an interval may start at any date between r_i and d_i . An interval is *accepted* if it is scheduled (without preemption) on one of the machines of the system at or after date r_i and before date d_i , during a time t_i such that $q_i \leq t_i \leq p_i$. When an interval is accepted from date t_1 to date t_2 , the machine is *busy* for all $t_1 \leq t < t_2$. An interval is said to be *degraded* if $t_i < p_i$, otherwise, it is *non-degraded*. Given a schedule O we denote by $N(O)$ its *size* (i.e. the number of scheduled intervals) and by $W(O)$ its *weight*, which is the sum of the $t_i \frac{w_i}{p_i}$'s over

all the accepted intervals. Given an interval instance σ let $N^*(\sigma)$ (resp. $W^*(\sigma)$) be the maximum number (resp. maximum weight) of intervals of σ which can be scheduled. In the sequel, if A is a polynomial-time algorithm, we denote by ρ_N (resp. ρ_W) the *approximation ratio for the size* (resp. weight), satisfying for all interval instances σ , $\rho_N N(A(\sigma)) \geq N^*(\sigma)$ (resp. $\rho_W W(A(\sigma)) \geq W^*(\sigma)$).

Related works. When degradation is not allowed, maximizing the size of an interval schedule can be done in polynomial time using the algorithms of Faigle et al. in [8]. Maximizing the weight of a non-degradable schedule can also be done in polynomial time using the algorithms of Carlisle et al. in [6] and Arkin et al. in [1]. Note that in the non-degradable intervals case, the size is a sub-problem of the weight (by letting all the weights to be one). Clearly, this is not the case for degradable intervals since a degraded interval could have a weight less than one unit. In task scheduling problems, there is a time window $[r_i, d_i]$ greater than the processing time p_i , in which each scheduled interval must be executed for exactly a time p_i . The problem of scheduling *non-degradable tasks* ($d_i \geq r_i + p_i$) has been studied by Bar-Noy et al. [5] and it is shown that both problems are NP-hard and approximable within a ratio near to 1.58 for the size problem and near to 2.54 for the weight problem. In [2], we studied the *simultaneous* maximization of the size and the weight. In [3] and [4], we studied the case where the intervals require more than one machine to be executed. We also analyzed in [3] the impact of the degradation of the number of machines allotted to intervals on the two criteria.

Our contribution. In Section 2, we show how the *immediate* and the *non-immediate* scheduling problems can be reduced to the non degradable intervals ($d_i = r_i + p_i$ and $q_i = p_i$) or tasks ($d_i \geq r_i + p_i$) scheduling problems for the size maximization. This implies that the immediate scheduling problem can be solved in polynomial-time and the non-immediate scheduling problem is NP-hard and can be approximated within a ratio of 1.58 with the algorithm of Bar-Noy et al [5].

In Section 3, we study the weight objective. We show that the *immediate scheduling problem* can be solved in polynomial-time. We present an algorithm, inspired by [6], exploiting the fact that the intervals finish their execution at special dates (either a release date or their own deadlines), which transforms the problem into finding the minimum cost flow of size k on an associated directed acyclic graph. For the non-immediate scheduling problem, we propose a 2.54-approximation algorithm for the special case where the weights are equal to the maximum execution duration for each intervals ($w_i = p_i$).

2 Size Maximization

In this section, we study the size maximization problem, i.e. the problem of maximizing the number of scheduled intervals. We show that the problem can be transformed into a non-degradable interval scheduling problem (solvable in

polynomial-time, see [1, 6, 8]) for the immediate model and into a task scheduling problem (NP-hard, see [5]) for the non-immediate model. All the results of this section are consequences of the following lemma which shows that for both variants there is an optimal solution executing each interval during the minimum amount of time. Note that since we are studying the size, we do not mention the weight w_i of the intervals in this section.

Lemma 1. *For the degradable interval scheduling problem, in both models, there exists an optimal schedule for the size in which all the scheduled intervals are fully degraded (every scheduled interval σ_i is executed for a time q_i).*

Proof. By contradiction, assume the existence of a schedule O^* with an optimal size having at least one of its intervals scheduled for a longer time than q_i and accepting more intervals than O_d , the schedule with maximal size in which all intervals are fully degraded. Let O'^* be the schedule obtained by degrading all the intervals of O^* . Clearly, O'^* is valid and accepts the same number of intervals as O^* so $N(O'^*) > N(O_d)$. This contradicts the optimality of O_d . \square

2.1 Immediate Scheduling Problem

We solve in polynomial time the size problem with this algorithm:

Transform the instance σ of degradable intervals $\sigma_i = (r_i, p_i, q_i, d_i)$ into an instance σ' of non-degradable intervals $\sigma'_i = (r'_i = r_i, p'_i = q_i, d'_i = r_i + q_i)$.
Apply an optimal algorithm (see [1, 6, 8]) on σ' .

Theorem 1. *For the immediate model, the above algorithm returns an optimal schedule for the size in polynomial-time.*

Proof. Optimality follows from Lemma 1 and the fact that the intervals have to start at their release dates. As the instance transformation is polynomial and all the cited algorithms are polynomial, our algorithm is also polynomial. \square

2.2 Non-immediate Scheduling Problem

Here, we show that maximizing the size in the non-immediate model is NP-hard because it contains the non-degradable task ($d_i \geq r_i + p_i$) scheduling problem. We show that any algorithm with an approximation ratio for the task scheduling problem is also an approximation algorithm for our problem. So, the algorithm of Bar-Noy et al. [5] is also a 1.58-approximation algorithm for this problem.

Theorem 2. *In the non-immediate model, maximizing the size is NP-hard.*

Proof. By Lemma 1, there is an optimal solution executing all the intervals for a time q_i . Therefore, one can transform any input instance I into an instance of tasks I_t by the following transformation: For each degradable interval $\sigma_i = (r_i, p_i, q_i, d_i)$ add the task $\sigma'_i = (r'_i = r_i, p'_i = q_i, d'_i = d_i)$ to I_t . The inverse

transformation (from any set of non-degradable tasks to a set of degradable intervals) can also be done easily. A schedule of I_t on k machines is then equivalent to a schedule of I on k machines. Since maximizing the size of a schedule for tasks is NP-hard (see [5]), our problem is also NP-hard. \square

Theorem 3. *For the size criterion, if A is an α -approximation algorithm for the task scheduling problem, then A is also an α -approximation for the non-immediate model. Consequently, a 1.58-approximation algorithm exists.*

Proof. Let σ be an instance of degradable intervals, that is a set of $\sigma_i = (r_i, p_i, q_i, d_i)$. Let σ' be the task schedule instance build from σ as follows: For each $\sigma_i = (r_i, p_i, q_i, d_i)$, add the task $\sigma'_i = (r'_i = r_i, p'_i = q_i, d'_i = d_i)$. From Lemma 1, we get $N^*(\sigma) = N^*(\sigma')$ and the first point of theorem follows.

Bar-Noy et al. in [5] approximate the size of task schedules within a ratio of 1.58. The ratio is then the same for the non-immediate scheduling problem. \square

3 Weight Maximization

Here, the maximization of the weight objective is studied. For the immediate model, we present in Section 3.1 an optimal algorithm. Indeed, we reduce the problem to a mincost flow problem, which is solvable in polynomial time (see [7] or [9]). In [6] and [1] such a reduction has been used in order to show that the non-degradable interval scheduling problem is polynomial. For the non-immediate model, we propose in Section 3.2 a 2-approximation algorithm for the single machine case and proportional weights and we extend it, using Theorem 3.3 of Bar-noy et al. [5], to a 2.54-approximation for the multiple machines case.

3.1 Immediate Scheduling Problem

We will exploit the following property showing that no gain can be obtained by creating idle times due to over-degradation.

Property 1 *Let σ be an instance of degradable intervals and O^* be a schedule of σ with an optimal weight in the immediate model. The completion time of any interval of O^* is either its own deadline or a release date of another scheduled interval.*

Proof. By contradiction, suppose the existence of an optimal schedule for the weight O^* such that an accepted interval σ_j is degraded and complete its execution at a date which is neither a release date nor its deadline. Let c_j be its completion time and σ_z be the interval that directly follows σ_j on the machine i where σ_j is executed. Let r_z be the release date of σ_z ; r_z is also the date at which σ_z starts its execution, thus $c_j \leq r_z$, and i is idle between these two dates. Two cases must be considered if σ_z exists: If $r_z \geq d_j$, let O'^* be the schedule executing the same intervals at the same dates as O^* except for σ_j which is enlarged until d_j . If $r_z < d_j$, let O'^* be the schedule executing the same intervals

at the same dates as O^* except for σ_j which is enlarged until r_z . If σ_z does not exist, let O'^* be the schedule executing the same intervals at the same dates as in O^* except for σ_j which is enlarged until d_j . In both cases, O'^* is valid and $W(O'^*) > W(O^*)$ because of the increase of the execution time of σ_j . This contradicts the optimality of O^* . \square

In spite of the infinity of execution alternatives of an interval, the decision points are deadlines and release dates. In the sequel, we call *significant* all schedules having for completion times of all the executed intervals either release dates or deadlines. Note that, by Property 1, all optimal schedules are significant.

We present now a polynomial time optimal algorithm for the weight maximization in the immediate model. We transform the problem into a mincost flow problem by constructing a directed graph from an interval instance. We show that this graph has a polynomial number of nodes. Then, we show that at every mincost flow of size k , we can associate a valid optimal significant schedule. To finish, we show how to build this schedule in polynomial time.

Transformation of an instance of intervals into a directed graph

Let σ be an instance of degradable intervals, $R = \{r_i : \sigma_i = (r_i, p_i, d_i, w_i) \in \sigma\}$ be the set of the release dates, and $\Omega_i = \{r_i\} \cup \{r : r \in R, r_i + q_i \leq r \leq d_i\} \cup \{d_i\}$ be the set of significant dates of $\sigma_i \in \sigma$.

Creation of graph $\mathcal{G}(\sigma) = (V, E)$, where V (resp. E) is the set of vertices (resp. arcs).

Vertices of $\mathcal{G}(\sigma)$: Each vertex u of V is associated to a date $date(u)$.

- Create vertices S and T with $date(S) = -\infty$ and $date(T) = +\infty$.
- **s-type vertices:** For each date $r \in R$, create a vertex s with $date(s) = r$. These $|R|$ vertices are indexed following the order of their associated dates (if $i < j$ then $date(s_i) < date(s_j)$).
- **u-type vertices:** For each interval σ_i and for each date d in Ω_i , create a new vertex associated with date d . These $|\Omega_i|$ vertices $u_{i,1}, u_{i,2}, \dots, u_{i,|\Omega_i|}$ are indexed following the order of their associated dates:
 $r_i = date(u_{i,1}) < \dots < date(u_{i,|\Omega_i|}) = d_i$.

Arcs of $\mathcal{G}(\sigma)$: Each arc is associated to a *capacity* and a *weight*.

- Create the $|R| + 1$ arcs $(S, s_1), (s_1, s_2), \dots, (s_{|R|}, T)$ all with capacity k and weight 0.
- For each interval σ_i create the following arcs:
 - For $j = 1, \dots, |\Omega_i| - 1$: Create the arcs $(u_{i,j}, u_{i,j+1})$ with capacity 1 and weight $-(date(u_{i,j+1}) - date(u_{i,j})) \frac{w_i}{p_i}$.
 - Create the arc $(s_\beta, u_{i,1})$ with capacity 1 and weight 0 with $1 \leq \beta \leq |R|$ and $date(s_\beta) = date(u_{i,1}) = r_i$.
 - If there exists a date r in R such that $d_i = date(u_{i,|\Omega_i|}) \leq r$, then let s_γ ($date(s_\gamma) \in R$) be the vertex such that $date(s_\gamma) = \min\{date(r) : d_i \leq date(r), r \in R\}$, and create the arc $(u_{i,|\Omega_i|}, s_\gamma)$ with capacity 1 and weight 0. Otherwise, add the arc $(u_{i,|\Omega_i|}, T)$ with capacity 1 and weight 0.
 - For $j = 2, 3, \dots, |\Omega_i| - 1$, create the arc $(u_{i,j}, s_\beta)$, $2 \leq \beta \leq |R|$ with $date(s_\beta) = date(u_{i,j})$ with capacity 1 and weight 0.

Let us show that the mincost flow of size k of any graph built by the transformation algorithm yields to an optimal k machines schedule for the weight, and that this can be done in polynomial time.

Lemma 2. *The number of nodes produced by the algorithm is $\mathcal{O}(n^2)$.*

Proof. The total number of nodes created by the algorithm is $2 + |R| + \sum_{\sigma_i \in \sigma} |\Omega_i|$, where $|R|$ is the number of distinct release dates and $\sum_{\sigma_i \in \sigma} |\Omega_i|$ represents the number of nodes created for the n intervals. The additive factor 2 comes from S and T . Note that $|R| \leq n$ because in the worst case, all intervals have distinct release dates. Let us upper bound $\sum_{\sigma_i \in \sigma} |\Omega_i|$, the number of nodes generated from the intervals. For each σ_i , $|\Omega_i|$ nodes are created. But $|\Omega_i| \leq |R| + 1$ and since there are n intervals, $\sum_{\sigma_i \in \sigma} |\Omega_i| \leq \mathcal{O}(n^2)$. The lemma follows. \square

Lemma 3. *The transformation algorithm returns an acyclic graph.*

Proof. Suppose by contradiction that there is a cycle in a graph produced by the algorithm. By construction, when an arc is added between two vertices u and v , we have $date(u) \leq date(v)$. Furthermore, for all intervals $\sigma_i \in \sigma$, we have $|\Omega_i| \geq 2$. By construction, there is no cycle containing only s -type vertices or containing only u -type vertices. If a cycle is possible in the graph, it contains both u -type and s -type vertices. No arc $(u_{i,j}, u_{k,l})$ where $i \neq k$ is present in the graph. Furthermore, to reach a u -type node from any s -type node one has to visit a node $u_{i,1}$. But this node cannot have an s -type node as a successor. Thus, the only possible cycles contain a path of at least two u -type nodes between two s -type nodes. Let $u_{i,1}$ and $u_{i,j}$, $j \geq 2$, be the extreme vertices of these u -type nodes path, and let s_α and s_β be respectively the predecessor of $u_{i,1}$ and the successor of $u_{i,j}$. By construction, we have $date(s_\alpha) = date(u_{i,1}) < date(u_{i,j}) = date(s_\beta)$. Thus since s_α is in a cycle there must exist an arc (X, s_α) with an elementary path from s_β to X . Thus, $date(X) < date(s_\alpha) < date(s_\beta) < date(X)$: A contradiction. \square

Given any path p from S to T , one can extract a valid significant schedule with the following algorithm:

Extraction of a schedule from a path

Let σ be an instance of partially degradable intervals.
 Let $\mathcal{G}(\sigma)$ be the graph constructed with the transformation algorithm.
 Let p be a path from S to T
 Let O be a current schedule, initialized to empty.
 For each arc (v, w) of p :
 if v and w are u -type vertices, $(v, w) = (u_{i,j}, u_{i,j+1})$, schedule in O the corresponding “piece of the interval” σ_i between $date(u_{i,j})$ and $date(u_{i,j+1})$
 While there are two consecutive “pieces of interval” σ_i in O , merge these two pieces.

Lemma 4. *This algorithm yields a valid significant schedule on one machine.*

Proof. Let O be the schedule returned by the extraction algorithm. In order to prove the validity of O , we have to show that each accepted interval σ_i has a processing time at least q_i and at most p_i and starts at r_i , secondly, all $\sigma_i \in O$ are executed continuously and finally, two scheduled intervals do not overlap.

By construction of $\mathcal{G}(\sigma)$, if σ_i is extracted, this means that p follows the arc $(s_\beta, u_{i,1})$ with $date(s_\beta) = date(u_{i,1})$. Then, p follows $(u_{i,1}, u_{i,2})$ thus the

algorithm executes the piece of interval σ_i between the dates $date(u_{i,1})$ and $date(u_{i,2})$. But, by definition, $date(u_{i,1}) = r_i$ and $date(u_{i,2}) = \min\{r \in R : r_i + q_i \leq r \leq d_i\}$. Thus, σ_i is executed at least during a time q_i . If p follows all $u_{i,j}$ nodes then O schedules σ_i for a processing time equal to p_i . Note that since $\mathcal{G}(\sigma)$ is acyclic (Lemma 3), no node $u_{i,j}$ can be visited twice by p . This means that no interval can be executed for a time larger than p_i . The only way for p to reach a node $u_{i,j}$ in $\mathcal{G}(\sigma)$ is to visit the node $u_{i,1}$. Thus, σ_i cannot be interrupted and executed later in O . The fact that there cannot be overlapping intervals follows from the fact that, by construction, all arcs (u, v) satisfy $date(u) \leq date(v)$.

It remains to show that O is significant. This is true because if an interval is executed, it starts at r_i (because p follows $u_{i,1}$). Furthermore, it finishes either at a release date if it is degraded or at its own deadline (each node $u_{i,j}$ is associated to a release date except the last one which is associated to d_i). \square

Lemma 5. *Let σ be an instance of degradable intervals, and $\mathcal{G}(\sigma)$ be the associated directed acyclic graph. From a mincost flow of size k (from S to T), we can extract an optimal schedule having a weight equal to the opposite of the total cost of the flow.*

Proof. Let us consider a mincost flow F of size k in $\mathcal{G}(\sigma)$. Since $\mathcal{G}(\sigma)$ is acyclic, each unit of flow can follow a particular elementary path from S to T .

Lemma 4 shows that for each unit of flow, we construct a valid significant schedule on a single machine. We have to show that the union of these one machine schedules is still valid on k machines and that the obtained schedule is optimal for the weight. For the validity, we just have to show that there is no interval scheduled twice on different machines. To make it possible 2 units of flow have to cross an arc (s_β, u_{i1}) . But this is not possible because the capacity of this arc is only one.

The weight of the corresponding schedule is the opposite of the cost of the flow. Assume by contradiction that there exists a significant schedule generating a weight greater than the opposite of the cost of F . Using the inverse technique of the extraction algorithm, one can find in $\mathcal{G}(\sigma)$, k elementary paths from S to T . By adding one unit of flow on each arc among these k paths, we get a flow of size k with a lower cost than F . Contradiction. Our technique constructs the optimal significant valid schedule. A consequence of Property 1, is that an optimal significant schedule is also optimal for the immediate model. \square

Theorem 4. *A schedule with an optimal weight in the immediate model can be found in polynomial-time.*

Proof. We just have to transform any instance of degradable intervals into a graph using our algorithm, to compute the mincost flow of size k on this graph with one of the algorithms described in [7] or in [9], and to extract the optimal schedule according to the flow. All these steps can be done in polynomial-time. Lemma 5 ensures the optimality. \square

3.2 Non-immediate Scheduling Problem for Proportional Weights

We study the maximization of the *weight* for the non-immediate model for the particular case where the weights are equal to the execution times ($w_i = p_i$). We describe a 2-approximation algorithm for a single machine system ($k = 1$). We extend it to a 2.54 approximation algorithm in the general case of k identical machines by a method used by Bar-Noy et al. in [5]. At this time we do not know whether the problem is NP-hard.

A 2-Approximation Algorithm for a Single Machine System. Before describing our results we need some definitions.

Definition 1. Let σ be an instance of degradable intervals. The **shadow** of σ is the set of dates (real numbers): $\text{Shadow}(\sigma) = \bigcup_{\sigma_i \in \sigma} \{x : r_i \leq x < d_i\}$.

We define the **thickness** $t_\sigma(x)$ of any date x to be the number of intervals σ_i of σ such that $r_i \leq x < d_i$: $t_\sigma(x) = |\{\sigma_i \in \sigma : r_i \leq x < d_i\}|$.

The thickness t_σ of an instance σ is the maximal thickness of $\text{shadow}(\sigma)$: $t_\sigma = \max\{t_\sigma(x) : x \in \text{Shadow}(\sigma)\}$.

Clearly, any valid schedule (on one machine) is a set of intervals whose shadow has thickness equal to one. The thickness of any instance is at least 1 (if no intervals overlap) and is at most $n = |\sigma|$.

The shadow is not necessarily composed by a single real interval. We call *component* each connected real sub-interval of the shadow, i.e. subsets S_i of the shadow, maximal for inclusion satisfying: $\forall x, y \in S_i$, if $x \leq z \leq y$ then $z \in S_i$.

Definition 2. Let σ be an instance of degradable intervals and S_i be any component of $\text{Shadow}(\sigma)$ that is a real interval of the form $[b_i, e_i)$. The weight of any component S_i is the value $e_i - b_i$ and the **weight of the shadow** is $W(\text{Shadow}(\sigma)) = \sum_{S_i} W(S_i)$.

Note that for any valid schedule O of σ we have: $W(\text{Shadow}(\sigma)) \geq W(O)$.

Lemma 6. Let σ be an instance of degradable intervals. There exists a set $\sigma' \subseteq \sigma$ such that: $t_{\sigma'} \leq 2$ and $\text{Shadow}(\sigma) = \text{Shadow}(\sigma')$.

Proof. Let us consider the following algorithm.

```

 $\sigma' := \sigma$ ;
While there exists a date  $x$  with  $t_{\sigma'}(x) > 2$  do
  Let  $x$  be a date with  $t_{\sigma'}(x) > 2$ ;
  Let  $C(x) = \{\sigma_i : \sigma_i \in \sigma' \text{ and } r_i \leq x < d_i\}$ 
  Let  $\sigma_a$  be the interval of  $C(x)$  having the minimum release date.
  Let  $\sigma_b$  be the interval of  $C(x)$  having the maximum deadline.
  Update  $\sigma'$  by removing all the intervals of  $C(x)$  except  $\sigma_a$  and  $\sigma_b$  (note that we may have  $\sigma_a = \sigma_b$ ).
```

This algorithm executes at most $n = |\sigma|$ times the While loop. Each execution strictly reduces the number of intervals of σ' . At the end, we clearly have $t_{\sigma'} \leq 2$. Let us now prove that: $\text{Shadow}(\sigma) = \text{Shadow}(\sigma')$. As $\sigma' \subseteq \sigma$ we have:

$Shadow(\sigma') \subseteq Shadow(\sigma)$. Assume by contradiction that there exists a date $y \in Shadow(\sigma)$ and $y \notin Shadow(\sigma')$. This means that there is a particular step in the main While loop, examining date x , before which $t_{\sigma'}(x) > 2$ and $y \in Shadow(\sigma')$ and after which $t_{\sigma'}(x) \leq 2$ and $y \notin Shadow(\sigma')$. This While loop examines the date x and removes from σ' all the intervals of $C(x)$ except σ_a and σ_b . By hypothesis, a consequence is that after the deletions we have: $t_{\sigma'}(y) = 0$. This means that y is neither a date of σ_a nor a date of σ_b . However, by the definition of σ_a and σ_b we have: $r_a \leq r_b < d_b$ but since $r_a < y < d_b$ and $d_a \leq y$ (otherwise y is a date of σ_a . Note that the date d_a is **not** a date of σ_a) and $y < r_b$ (otherwise y is a date of σ_b) we get: $r_a < d_a \leq y < r_b < d_b$. This implies that σ_a and σ_b do not intersect and contradicts the fact that σ_a and σ_b must have the date x in common. The contradiction follows. \square

Let σ be any degradable intervals instance. Each degradable interval $\sigma_i = (r_i, d_i, p_i, q_i, w_i = p_i) \in \sigma$ is transformed into a non degradable interval: $\sigma_i^N = (r_i, d_i, p_i, q_i = p_i, w_i = p_i)$. The set of these non degradable intervals is denoted σ^N . Let A be the algorithm described in [1] (or in [6]) that constructs in polynomial time an optimal weight schedule from any instance of non degradable intervals ($q_i = p_i$). Our algorithm ADN just consists in applying algorithm A to σ^N . Note that the obtained schedule is valid and that the whole algorithm is polynomial. The next result shows that ADN is a 2-approximation algorithm.

Theorem 5. *Let σ be an instance of degradable intervals, $k = 1$, O^* be a maximal weight degradable schedule of σ and σ^N be the non degradable instance associated to σ . Then O , the resulting schedule of A on σ^N , satisfies: $W(O) \geq \frac{W(O^*)}{2}$.*

Proof. By Lemma 6, there is a set $\sigma' \subseteq \sigma$ such that: $t_{\sigma'} \leq 2$ and $Shadow(\sigma) = Shadow(\sigma')$. σ' can be partitioned into 2 subsets: $\sigma'(1)$ and $\sigma'(2)$ with $t_{\sigma'(1)} = t_{\sigma'(2)} = 1$. This means that all the intervals of $\sigma'(1)$ (resp. $\sigma'(2)$) can be scheduled without degradation (between their release dates and their deadlines) on one machine. The sets $\sigma'(1)$ and $\sigma'(2)$ can be considered as valid non degradable schedules of σ' . W.l.o.g. assume that $W(\sigma'(1)) \geq W(\sigma'(2))$.

Since A constructs an optimal weight non degradable schedule of σ^N and as $\sigma'(1) \subseteq \sigma^N$, we have: $W(O) \geq W(\sigma'(1))$. Moreover, since $W(\sigma'(1)) \geq W(\sigma'(2))$, we have: $W(\sigma'(1)) \geq \frac{W(\sigma'(1)) + W(\sigma'(2))}{2}$. But, $\frac{W(\sigma'(1)) + W(\sigma'(2))}{2} \geq \frac{W(Shadow(\sigma'))}{2}$. As σ' satisfies: $Shadow(\sigma) = Shadow(\sigma')$ (by Lemma 6), we deduce: $\frac{W(Shadow(\sigma'))}{2} = \frac{W(Shadow(\sigma))}{2}$. Finally, since $W(Shadow(\sigma)) \geq W(O^*)$, we have: $W(O) \geq \frac{W(O^*)}{2}$.

Note that the bound is tight. Indeed, consider the following 4 intervals instance: $\sigma_1 = (r_1 = 0, p_1 = 1, d_1 = 1, q_1 = 1 - \varepsilon, w_1 = 1)$, $\sigma_2 = (r_2 = 1 - \varepsilon, p_2 = 1 + 2\varepsilon, d_2 = 2 + \varepsilon, q_2 = 2, w_2 = 1 + 2\varepsilon)$, $\sigma_3 = (r_3 = 2, p_3 = 1, d_3 = 3, q_3 = 1 - \varepsilon, w_3 = 1)$, $\sigma_4 = (r_4 = 3 - \varepsilon, p_4 = 1 + 2\varepsilon, d_4 = 4 + \varepsilon, q_4 = 1 + 2\varepsilon, w_4 = 1 + 2\varepsilon)$. A schedules σ_2 and σ_4 generating a weight of $2 + 4\varepsilon$. The optimal schedule consists in scheduling all the intervals degrading σ_1 and σ_3 generating a weight of $4 + \varepsilon$. $\frac{4 + \varepsilon}{2 + 4\varepsilon}$ tends to 2 when ε tends to 0. \square

Multiple Machines Systems. We propose an algorithm $\mathcal{A}(k)$ for finding a schedule from an instance of degradable intervals σ for k machines. $\mathcal{A}(k)$ uses as a subroutine ADN. Using similar arguments than Bar-Noy et al. in [5], we get Theorem 6.

Algorithm $\mathcal{A}(k)$

Input: Any degradable intervals instance σ and k identical machines.
 For $i := 1$ to k do
 Apply ADN on instance σ and machine number i ;
 Remove from σ the previously scheduled intervals;
 Return the schedule composed of the k constructed sub-schedules;

Theorem 6. *For the degradable intervals scheduling problem on k identical machines, $\mathcal{A}(k)$ is a $\frac{(2k)^k}{(2k)^k - (2k-1)^k}$ -approximation. Note that $\frac{(2k)^k}{(2k)^k - (2k-1)^k} \leq 2.54$.*

Final Discussion and Future Work. The maximization of the size and the weight in the immediate model can be done in polynomial time. The size maximization in the non-immediate model, is NP-Hard and the complexity of the weight problem is unknown. Nevertheless, we have proposed approximation algorithms for both metrics. In the future, we will focus on the NP-hardness of the weight problem in the non-immediate model. We will also seek to generalize the results presented here for the weight maximization in the non-immediate model to the case where the weights can take any values.

References

1. Arkin, E., Silverberg, B.: Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics* **18** (1987) 1–8
2. Baille, F., Bampis, E., Laforest, C.: A note on optimal bicriteria approximation schedules for simultaneous maximization of weight and profit. PPL, to appear (2004)
3. Baille, F., Bampis, E., Laforest, C.: Bicriteria scheduling of parallel degradable tasks for network access under pricing constraints. INOC 2003 proceedings (2003) 37–42
4. Baille, F., Bampis, E., Laforest, C.: Ordonnancements de tâches parallèles avec compromis entre la taille et le poids (in french). EARO (2003)
5. Bar-Noy, A., Guha, S., Naor, J., Schieber, B.: Approximating the throughput of multiple machines under real-time scheduling. STOCS 99 (1999) 622–631
6. Carlisle, Martin C., Lloyd, Errol L.: On the k -coloring of intervals. *Discrete Applied Mathematics* **58** (1995) 225–235
7. Jack Edmonds, Richard M. Karp: Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM* **19** (1972) 248–264
8. Faigle, U., Nawijn, M.: Note on scheduling intervals on-line. *Discrete Applied Mathematics* **58** (1995) 13–17
9. Robert Endre Tarjan: Data structures and network algorithms. Society for Industrial and Applied Mathematics (1983)

Minimizing Maximum Lateness on Identical Parallel Batch Processing Machines

Shuguang Li^{1,*}, Guojun Li^{2,3, **}, and Shaoqiang Zhang⁴

¹ Department of Mathematics and Information Science, Yantai University,
Yantai 264005, P.R. China

sgliytu@hotmail.com

² Institute of Software, Chinese Academy of Sciences, Beijing 100080, P.R. China

³ School of Mathematics and System Science, Shandong University,
Jinan 250100, P.R. China

gjli@sdu.edu.cn

⁴ Mathematics Department, Tianjin Normal University,
Tianjin 300074, P. R. China

sqzhang@163.com

Abstract. We consider the problem of scheduling n jobs with release dates on m identical parallel batch processing machines so as to minimize the maximum lateness. Each batch processing machine can process up to B ($B < n$) jobs simultaneously as a batch, and the processing time of a batch is the largest processing time among the jobs in the batch. Jobs processed in the same batch start and complete at the same time. We present a *polynomial time approximation scheme* (PTAS) for this problem.

1 Introduction

In this paper, we consider the following batch scheduling problem that arises in the semiconductor industry. The input to the problem consists of n jobs and m identical batch processing machines that operate in parallel. Each job j has a processing time p_j , which specifies the minimum time needed to process the job without interruption on any one of the m batch processing machines. In addition, each job j has a release date r_j before which it cannot be processed and a delivery time q_j . Each job's delivery begins immediately after its processing has been completed, and all jobs may be delivered simultaneously. Each batch processing machine can process up to B ($B < n$) jobs simultaneously as a batch. The processing time of a batch is the largest processing time among the jobs in the batch. The completion time of a batch is equal to its start time plus its processing time. Jobs processed in the same batch have the same completion time, which is the completion time of the batch in which the jobs contained.

* Corresponding author

** This work was supported by the fund from NSFC under fund numbers 10271065 and 60373025.

This model is called the *bounded model*, which is motivated by the problem of scheduling burn-in operations in the manufacturing of integrated circuit (IC) chips (See [1] for the detailed process). Our goal is to schedule the jobs on m batch processing machines so as to minimize $\max_j \{C_j + q_j\}$, where C_j denotes the completion time of job j in the schedule.

The problem as stated is equivalent to that with release dates and due dates, d_j , rather than delivery times, in which case the objective is to minimize the maximum lateness, $L_j = C_j - d_j$, of any job j . When considering the performance of approximation algorithms, the delivery-time model is preferable (see [2]). Because of this equivalence, we denote the problem as $P|r_j, B|L_{\max}$, using the notation of Graham et al. [3].

Problems related to scheduling batch processing machines have been examined extensively in the deterministic scheduling literature in recent years. Here, we give a brief survey on the previous work on the bounded problems with due dates. Lee et al. [1] presented a heuristic for the problem of minimizing maximum lateness on identical parallel batch processing machines and a worst-case ratio on its performance. They also provided efficient algorithms for minimizing the number of tardy jobs and maximum tardiness under a number of assumptions. Brucker et al. [4] summarized the complexity of the problems of scheduling a batch processing machine to minimize regular scheduling criteria that are non-decreasing in the job completion times. Along with other results, they proved that the bounded problems of minimizing the maximum lateness, the number of tardy jobs and the total tardiness on a single batch processing machine are strongly *NP*-hard, even if $B = 2$. Both [1] and [4] concentrated on the model of equal release dates. As for the model of unequal release dates, the known previous results were restricted on the case of a single batch processing machine [5–7]. Ikura and Gimple [5] provided an $O(n^2)$ algorithm to determine whether a due date feasible schedule exists under the assumption that release dates and due dates are agreeable (i.e., $r_i \leq r_j$ implies $d_i \leq d_j$) and all jobs have identical processing times. Li and Lee [6] proved that the problems of minimizing the maximum tardiness and the number of tardy jobs where all jobs have identical processing times are strongly *NP*-hard even if release dates and due dates are agreeable. Wang et al. [7] presented a genetic algorithm to minimize the maximum lateness with release dates on a single batch processing machine.

To the best of our knowledge, the general $P|r_j, B|L_{\max}$ problem has not been studied to date. In this paper we present a PTAS for this problem. Our study has been initiated by [8] and [9]. Deng et al. [8] presented a PTAS for the more vexing problem of minimizing the total completion time with release dates on a single batch processing machine. Hall and Shmoys [9] presented a PTAS for problem $P|r_j|L_{\max}$ (the special case of our problem where $B = 1$). We draw upon several ideas from [8, 9] to solve our problem.

This paper is organized as follows. In Section 2, we introduce notations, simplify our problem by applying the rounding method, and define small and large jobs. In Section 3, we describe how to batch the small jobs. In Section 4, we first show that there exists a $(1 + 5\epsilon)$ -approximate outline, a set of information

with which we can construct a $(1 + 5\epsilon)$ -approximate schedule. We proceed to define the concept of *outline* and then enumerate over all outlines in polynomial time, among which there is a $(1 + 5\epsilon)$ -approximate outline. Put together, these elements give us our PTAS for the general problem.

2 Preliminaries

We use opt to denote the objective value of the optimal schedule. To establish a *polynomial time approximation scheme* (PTAS), for any given positive number ϵ , we need find a solution with value at most $(1 + \epsilon) \cdot opt$ in time polynomial in the input size.

In this section, we aim to transform any input into one with simple structure. Each transformation potentially increases the objective function value by $O(\epsilon) \cdot opt$, so we can perform a constant number of them while still staying within a $1 + O(\epsilon)$ factor of the original optimum. When we describe such a transformation, we shall say that it produces $1 + O(\epsilon)$ *loss*. To simplify notations we will assume throughout the paper that $1/\epsilon$ is integral. For explicitness, we define the delivery time of a batch to be the largest delivery time among the jobs in the batch. We use $p(B_i)$, $d(B_i)$, $S(B_i)$ and $C(B_i)$ to denote the processing time, delivery time, start time and completion time respectively of batch B_i . We use $L_{\max}(S)$ to denote the objective value of schedule S . We call a batch containing exactly B jobs a *full* batch, where B is the batch capacity. A batch that is not full will be called a *partial* batch. Let $r_{\max} = \max_j r_j$, $p_{\max} = \max_j p_j$, $q_{\max} = \max_j q_j$.

The special case of $P|r_j, B|L_{\max}$ where all $r_j = q_j = 0$, denoted as $P|B|C_{\max}$, is already strongly *NP*-hard [1] (even for $B = 1$). Lee et al. [1] observed that there exists an optimal schedule for $P|B|C_{\max}$ in which all jobs are pre-assigned into batches according to the BLPT rule: rank the jobs in non-increasing order of processing times, and then batch the jobs by successively placing the B (or as many as possible) jobs with the largest processing times into the same batch.

To solve the general version of the problem with release dates, we need to amend the BLPT rule: we may use this rule for all the jobs even though they are not released simultaneously, or use it only for a subset of the jobs.

We use the BLPT rule for all the jobs and get a number of batches. Denote by d the total processing time of these batches. Then we have the following lemma.

Lemma 1. $\max\{r_{\max}, p_{\max}, q_{\max}, \frac{d}{m}\} \leq opt \leq r_{\max} + p_{\max} + q_{\max} + \frac{d}{m}$.

Proof. It's easy to see that $opt \geq \max\{r_{\max}, p_{\max}, q_{\max}, d/m\}$. We use the BLPT rule for all the jobs and get a number of batches. Starting from time r_{\max} we schedule these batches by *List Scheduling* algorithm [10]: whenever a machine is idle, choose any available batch to start processing on that machine. Any job will be delivered by time $r_{\max} + p_{\max} + q_{\max} + d/m$ in this schedule. \square

Let $\delta = \epsilon \cdot \max\{r_{\max}, p_{\max}, q_{\max}, d/m\}$. We will use δ this way throughout this paper.

A technique used by Hall and Shmoys [9] allows us to deal with only a constant number of distinct release dates and delivery times. The idea is to

round each release date and delivery time down to the nearest multiple of δ . Since $r_{\max} \leq (1/\epsilon)\delta$ and $q_{\max} \leq (1/\epsilon)\delta$, there are now at most $1/\epsilon + 1$ distinct release dates, as well as $1/\epsilon + 1$ distinct delivery times. Clearly, the optimal value of this transformed instance cannot be greater than opt . Every feasible solution to the modified instance can be transformed into a feasible solution to the original instance just by adding δ to each job's start time, and reintroducing the original delivery time. Since $\delta \leq \epsilon \cdot opt$, the solution value may increase by at most a $1 + 2\epsilon$ factor. Therefore we get the following lemma.

Lemma 2. *With $1 + 2\epsilon$ loss, we assume that there are at most $1/\epsilon + 1$ distinct release dates and $1/\epsilon + 1$ distinct delivery times.*

As a result of Lemma 2, we assume without loss of generality that the release dates take on $1/\epsilon + 1$ values, which we denote by $\rho_1, \rho_2, \dots, \rho_{1/\epsilon+1}$, where $\rho_i = (i - 1)\delta$. We set $\rho_{1/\epsilon+2} = \infty$. We partition the time interval $[0, \infty)$ into $1/\epsilon + 1$ disjoint intervals of the form $I_i = [\rho_i, \rho_{i+1})$ ($i = 1, 2, \dots, 1/\epsilon + 1$). We assume that the delivery times take on $1/\epsilon + 1$ values, which we denote by $\xi_1 < \xi_2 < \dots < \xi_{1/\epsilon+1}$.

We partition the jobs (batches) into two sets according to their processing times. We say that a job or a batch is *small* if its processing time is less than $\delta/(1/\epsilon + 1)^2$, and *large* otherwise. We use T_{il} to denote the set of small jobs with ρ_i and ξ_l as their common release date and delivery time respectively, $i = 1, 2, \dots, 1/\epsilon + 1$; $l = 1, 2, \dots, 1/\epsilon + 1$.

By Lemma 1, we know that there are at most $4(1/\epsilon + 1)^2/\epsilon$ large batches processed on each machine in any optimal schedule. More precisely, on each machine, there are at most $(1/\epsilon + 1)^2$ large batches started in interval I_i for $i = 1, 2, \dots, 1/\epsilon$ and at most $3(1/\epsilon + 1)^2/\epsilon$ large batches started in interval $I_{1/\epsilon+1}$. Though simple, this observation plays an important role in our algorithm since it allows us to show that the number of distinct outlines is polynomial in the input size. We defer the details till Section 4.

The following lemma enables us to deal with only a constant number of distinct processing times of large jobs.

Lemma 3. *With $1 + \epsilon$ loss, the number of distinct processing times of large jobs, κ , can be up-bounded by $4(1 + \epsilon)^2/\epsilon^4$.*

Proof. We round each large job's processing time down to the nearest multiple of $\epsilon/4 \cdot \delta/(1/\epsilon + 1)^2$. Clearly, the optimal value of the rounded instance cannot be greater than opt . Since $p_j \geq \delta/(1/\epsilon + 1)^2$ for each large job j , by replacing the rounded values with the original ones we can transform any optimal schedule for the rounded instance into a $(1 + \epsilon/4)$ -approximate schedule for the original instance. Since $p_j \leq (1/\epsilon)\delta$, we get $\kappa < 4(1 + \epsilon)^2/\epsilon^4$, as claimed. \square

Let $P_1 < P_2 < \dots < P_\kappa$ be the κ distinct processing times of large jobs. We assume henceforth that the original problem has the properties described in Lemmas 2 and 3.

3 Batching the Small Jobs

In this section, we describe how to batch the small jobs with $1 + 3\epsilon$ loss. The basic idea on which we rely is similar to that in [8].

For $i = 1, 2, \dots, 1/\epsilon + 1$ and $l = 1, 2, \dots, 1/\epsilon + 1$, we use the BLPT rule for all the jobs in T_{il} and get a number of batches, $B_{il}^1, B_{il}^2, \dots, B_{il}^{k_{il}}$, such that $B_{il}^1, B_{il}^2, \dots, B_{il}^{k_{il}-1}$ are full batches and $q(B_{il}^h) \geq p(B_{il}^{h+1})$, where $p(B_{il}^h)$ and $q(B_{il}^h)$ denote the processing times of the largest and smallest jobs in B_{il}^h , respectively. Then we have the following observation.

$$\sum_{h=1}^{k_{il}-1} (p(B_{il}^h) - q(B_{il}^h)) + p(B_{il}^{k_{il}}) < \delta / (1/\epsilon + 1)^2 \quad (1)$$

Let $\mathcal{B}_{il} = \{B_{il}^h : h = 1, 2, \dots, k_{il}\}$. We modify \mathcal{B}_{il} to define a new set of batches $\widetilde{\mathcal{B}}_{il} = \{\widetilde{B}_{il}^h : h = 1, 2, \dots, k_{il}\}$, where \widetilde{B}_{il}^h is obtained by letting all processing times in B_{il}^h be equal to $q(B_{il}^h)$ for $h = 1, 2, \dots, k_{il}-1$ and $\widetilde{B}_{il}^{k_{il}}$ is obtained by letting all processing times in $B_{il}^{k_{il}}$ be equal to zero.

Each original small job j is now modified to a new small job j' , with p'_j (which is equal to the processing time of the batch that contains j'), r_j and q_j as its processing time, release date and delivery time, respectively. We call the jobs $\{p'_j, r_j, q_j : j = 1, 2, \dots, n\}$ *modified jobs*, where $p'_j = p_j$ if j is a large job. We then define an accessory problem:

BS1: Schedule the modified jobs $\{p'_j, r_j, q_j : j = 1, 2, \dots, n\}$ on m identical parallel batch processing machines to minimize the maximum lateness.

We use $opt1$ to denote the optimal value to BS1. Since $p'_j \leq p_j$ for all jobs j , we get $opt1 \leq opt$. We also observe the following fact. (Please contact the authors for the detailed proof if any reader is interested in it.)

Lemma 4. *There exists a schedule S for BS1 with the objective value $L_{\max}(S) \leq opt1 + 2\epsilon \cdot opt$ in which the set of the batches containing small jobs is just $\bigcup_{i=1}^{1/\epsilon+1} \left(\bigcup_{l=1}^{1/\epsilon+1} \widetilde{\mathcal{B}}_{il} \right)$.*

The following lemma enables us to determine the batch structure of all the small jobs a priori.

Lemma 5. *There exists a $(1+3\epsilon)$ -approximate schedule S for $P|r_j, B|L_{\max}$ with the following properties:*

- 1) *the set of the batches in S containing small jobs is just $\bigcup_{i=1}^{1/\epsilon+1} \left(\bigcup_{l=1}^{1/\epsilon+1} \mathcal{B}_{il} \right)$;*
- 2) *the batches in S started in each interval on each machine are processed successively in the order of non-increasing delivery times.*

Proof. Consider a schedule for BS1 that is described in Lemma 4. We transform it by replacing the modified jobs with the original jobs. By the previous inequalities (1), we have $\sum_{h=1}^{k_{il}-1} (p(B_{il}^h) - p(\widetilde{B}_{il}^h)) + (p(B_{il}^{k_{il}}) - 0) < \delta / (1/\epsilon + 1)^2$

($i = 1, 2, \dots, 1/\epsilon + 1$; $l = 1, 2, \dots, 1/\epsilon + 1$). Thus the solution value may increase by $\delta \leq \epsilon \cdot \text{opt}$. As $\text{opt1} \leq \text{opt}$, we get a $(1 + 3\epsilon)$ -approximate schedule for $P|r_j, B|L_{\max}$ in which the set of the batches containing small jobs is just $\bigcup_{i=1}^{1/\epsilon+1} \left(\bigcup_{l=1}^{1/\epsilon+1} \mathcal{B}_{il} \right)$. We then use the well-known Jackson's rule [11] to transform this schedule without increasing the objective value: process successively the batches started in each interval on each machine in the order of non-increasing delivery times. Thus we get a schedule of the required type for $P|r_j, B|L_{\max}$. \square

4 Scheduling the Jobs

In this section we will design a polynomial time approximation scheme to solve problem $P|r_j, B|L_{\max}$.

By Lemma 5, we can pre-assign all the small jobs into batches and get $\bigcup_{i=1}^{1/\epsilon+1} \left(\bigcup_{l=1}^{1/\epsilon+1} \mathcal{B}_{il} \right)$ in $O(n \log n)$ time: use the BLPT rule for all the jobs in T_{il} ($i = 1, 2, \dots, 1/\epsilon + 1$; $l = 1, 2, \dots, 1/\epsilon + 1$). Thus we will restrict our attention to schedules of this form.

Next we show that there exists a $(1 + 5\epsilon)$ -approximate outline, a set of information with which we can construct a $(1 + 5\epsilon)$ -approximate schedule. We motivate the concept from [9].

Let us fix a $(1 + 3\epsilon)$ -approximate schedule that is described in Lemma 5, S . Let x_{ijl} be the total processing time of the small batches in S that are started in interval I_i on machine M_j with ξ_l as their common delivery time. Let $X_{ijl} = \lceil (1/\epsilon + 1)^2 \cdot x_{ijl} / \delta \rceil \cdot \delta / (1/\epsilon + 1)^2$. That is, X_{ijl} is the approximate amount of time in interval I_i on machine M_j that is spent processing the small batches with delivery time ξ_l . We then delete from S all the jobs and the small batches, but retain all the empty large batches. Let Y_{ijkl} be the set of the empty large batches in S that are started in interval I_i on machine M_j with P_k and ξ_l as their common processing time and delivery time respectively. Recall that P_k denotes the k th largest processing time among κ ($\kappa < 4(1 + \epsilon)^2/\epsilon^4$) distinct processing times of large jobs. We call the set $\{(X_{ijl}, Y_{ijkl}) : 1 \leq i \leq 1/\epsilon + 1, 1 \leq j \leq m, 1 \leq k \leq \kappa, 1 \leq l \leq 1/\epsilon + 1\}$ a $(1 + 5\epsilon)$ -approximate outline, since it can be used to construct a $(1 + 5\epsilon)$ -approximate schedule, as Lemma 6 below shows.

Algorithm A_1

Given a $(1 + 5\epsilon)$ -approximate outline, do the following:

- Step 1. Suppose that we have assigned small batches to intervals I_1 to I_{i-1} . We describe the assignment of small batches to I_i . ($i = 1, 2, \dots, 1/\epsilon + 1$.) For $j = 1, 2, \dots, m$ and $l = 1, 2, \dots, 1/\epsilon + 1$, we construct a set of small batches Z_{ijl} as follows: assign the small batches available in I_i with delivery time ξ_l to Z_{ijl} , until the first time that $\sum_{B_h \in Z_{ijl}} p(B_h) \geq X_{ijl}$ (or until there are no more available small batches with delivery time ξ_l). Clearly, if $X_{ijl} = 0$, then $Z_{ijl} = \phi$.

- Step 2. For $i = 1, 2, \dots, 1/\epsilon + 1$, $j = 1, 2, \dots, m$, we stretch I_i to make an extra space with length $2\delta/(1/\epsilon + 1)$ and then start the small batches in $\bigcup_{l=1}^{1/\epsilon+1} Z_{ijl}$ and the empty large batches in $\bigcup_{k=1}^{\kappa} \left(\bigcup_{l=1}^{1/\epsilon+1} Y_{ijkl} \right)$ as early as possible in I_i on M_j in the order of non-increasing delivery times (by Lemma 5).
- Step 3. For ease of representation, we reindex arbitrarily the large jobs as x_1, x_2, \dots . Index the empty large batches arbitrarily as B_1, B_2, \dots . By regarding each large job x_g as a vertex in X , and each empty large batch B_h as B (the batch capacity) vertices $y_{h1}, y_{h2}, \dots, y_{hB}$ in Y , we construct a bipartite graph G with bipartition (X, Y) , where x_g is joined to $y_{h1}, y_{h2}, \dots, y_{hB}$ if and only if $r_{x_g} \leq S(B_h)$, $p_{x_g} \leq p(B_h)$ and $q_{x_g} \leq d(B_h)$. Then we use the *Hungarian method* [12] to get a matching of G that saturates every vertex in X , which corresponds to a feasible assignment of the large jobs to the empty large batches.

Lemma 6. *Given a $(1 + 5\epsilon)$ -approximate outline, Algorithm A_1 will find a $(1 + 5\epsilon)$ -approximate schedule in $O((1/\epsilon + 1)^6 m^3 B^3 / \epsilon^3)$ time.*

Proof. Denote by S the $(1 + 3\epsilon)$ -approximate schedule from which the $(1 + 5\epsilon)$ -approximate outline is obtained. We will show that Algorithm A_1 reconstructs S with $1 + 2\epsilon$ loss in $O((1/\epsilon + 1)^6 m^3 B^3 / \epsilon^3)$ time.

Clearly, the definition of X_{ijl} , plus the way to assign small batches to intervals, guarantees that every small batch gets assigned to some set Z_{ijl} . We observe that for any i, j and l , $\sum_{B_h \in Z_{ijl}} p(B_h) - x_{ijl} < 2\delta/(1/\epsilon + 1)^2$. We also observe that there exists a matching of G that saturates every vertex in X (i.e., the assignment of the large jobs to the large batches in S). Therefore Algorithm A_1 constructs a feasible schedule S' with $L_{\max}(S') \leq L_{\max}(S) + 2\delta \leq (1 + 5\epsilon) \cdot \text{opt}$. Noting that a $(1 + 5\epsilon)$ -approximate outline contains at most $4(1/\epsilon + 1)^2 m/\epsilon$ empty large batches, we get $|X| \leq |Y| \leq 4(1/\epsilon + 1)^2 mB/\epsilon$. Therefore it will take $O((1/\epsilon + 1)^6 m^3 B^3 / \epsilon^3)$ time to get a matching of G that saturates every vertex in X . Since the time complexity of Algorithm A_1 is determined by Step 3, the lemma is proved. \square

A possibility of a $(1 + 5\epsilon)$ -approximate outline is called an *outline*. We are going to enumerate over all outlines in polynomial time, among which there is a $(1 + 5\epsilon)$ -approximate outline.

To do this, we define a *machine configuration* to be the restriction of an outline to a particular machine. Let us fix a particular machine, M_j . Recall that in any optimal schedule, on each machine, there are at most $(1/\epsilon + 1)^2$ large batches started in interval I_i for $i = 1, 2, \dots, 1/\epsilon$ and at most $3(1/\epsilon + 1)^2/\epsilon$ large batches started in interval $I_{1/\epsilon+1}$. Combining the structure of an outline, we get the following observation. For $i = 1, 2, \dots, 1/\epsilon$, $k = 1, 2, \dots, \kappa$ and $l = 1, 2, \dots, 1/\epsilon$, both the number of different possibilities of X_{ijl} and that of Y_{ijkl} are $(1/\epsilon + 1)^2 + 1$; while for $i = 1/\epsilon + 1$, $k = 1, 2, \dots, \kappa$ and $l = 1, 2, \dots, 1/\epsilon$, both the number of different possibilities of X_{ijl} and that of Y_{ijkl} are $3(1/\epsilon + 1)^2/\epsilon + 1$.

Therefore we can up-bound the number of different machine configurations by $\Gamma < \{[(1/\epsilon + 1)^2 + 1]^{1/\epsilon+1} \cdot [(1/\epsilon + 1)^2 + 1]^{(1/\epsilon+1)\kappa}\}^{1/\epsilon} \cdot \{[3(1/\epsilon + 1)^2/\epsilon + 1]^{1/\epsilon+1} \cdot [3(1/\epsilon+1)^2/\epsilon+1]^{(1/\epsilon+1)\kappa}\} < 2[(1/\epsilon+1)^2+1]^{(1/\epsilon+1)(\kappa+1)/\epsilon}$, where $\kappa < 4(1+\epsilon)^2/\epsilon^4$.

We denote the different machine configurations as $1, 2, \dots, \Gamma$. An outline can now be defined as a tuple $(m_1, m_2, \dots, m_\Gamma)$, where m_i is the number of machines with configuration i . Therefore there are at most $(m+1)^\Gamma$ outlines to consider, a polynomial in m .

Given an outline, we can evaluate its objective value as follows. View each X_{ijl} as an aggregated batch with processing time X_{ijl} and delivery time ξ_l respectively. For $i = 1, 2, \dots, 1/\epsilon + 1$ and $j = 1, 2, \dots, m$, we stretch I_i to make an extra space with length $\delta/(1/\epsilon + 1)$ and then start the aggregated batches $X_{ij1}, X_{ij2}, \dots, X_{ij(1/\epsilon+1)}$ and the empty large batches in $\bigcup_{k=1}^{\kappa} \left(\bigcup_{l=1}^{1/\epsilon+1} Y_{ijkl} \right)$ as

early as possible in I_i on M_j in the order of non-increasing delivery times. The time by which all batches have been delivered is the objective value of the given outline. If some batch (an aggregated batch or an empty large batch) cannot be started in the specified interval, then the outline will be excluded.

We are now ready to describe our algorithm, which constructs a feasible schedule from an outline with objective value as small as possible.

Algorithm A_2

- Step 1. Get all outlines, evaluate their objective values and exclude some of them as described above.
- Step 2. Invoke Algorithm A_1 repeatedly to deal with the left outlines in the order of non-decreasing objective values until a feasible schedule is generated. (If some small batch or a large job cannot be scheduled and has to be eventually left, then the outline cannot generate a feasible schedule and will be excluded.)
- Step 3. Clean up the generated feasible schedule (delete the empty batches and move all the batches to the left as far as possible while keep them in the specified intervals) and then output it.

Finally, we get the following theorem.

Theorem 1. *Algorithm A_2 is a PTAS for the problem $P|r_j, B|L_{\max}$.*

References

1. C. Y. Lee, R. Uzsoy, and L. A. Martin Vega: Efficient algorithms for scheduling semiconductor burn-in operations. *Operations Research* 40 (1992) 764–775
2. H. Kise, T. Ibaraki, and H. Mine: Performance analysis of six approximation algorithms for the one-machine maximum lateness scheduling problem with ready times. *Journal of the Operations Research Society of Japan* 22 (1979) 205–224
3. R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan: Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* 5 (1979) 287–326

4. P. Brucker, A. Gladky, H. Hoogeveen, M. Y. Kovalyov, C. N. Potts, T. Tautenhahn, and S. L. van de Velde: Scheduling a batching machine. *Journal of Scheduling* 1 (1998) 31–54
5. Y. Ikura and M. Gimple: Scheduling algorithms for a single batch processing machine. *Operations Research Letters* 5 (1986) 61–65
6. C. L. Li and C. Y. Lee: Scheduling with agreeable release times and due dates on a batch processing machine. *European Journal of Operational Research* 96 (1997) 564–569
7. C. S. Wang and R. Uzsoy: A genetic algorithm to minimize maximum lateness on a batch processing machine. *Computers and Operations Research* 29 (2002) 1621–1640
8. X. Deng, H. D. Feng, and G. J. Li: A PTAS for semiconductor burn-in scheduling. Submitted to *Journal of Combinatorial Optimization*.
9. L. A. Hall and D. B. Shmoys: Approximation schemes for constrained scheduling problems. *Proceedings of the 30th annual IEEE Symposium on Foundations of Computer Science* (1989) 134–139.
10. R.L.Graham: Bounds for certain multiprocessor anomalies. *Bell System Technical Journal* 45 (1966) 1563–1581
11. J. R. Jackson: Scheduling a production line to minimize maximum tardiness. Research Report 43, Management Science Research Project, UCLA, 1955.
12. J. A. Bondy and U. S. R. Murty: *Graph theory with applications*. Macmillan Press, 1976.

Efficient Algorithms for Approximating a Multi-dimensional Voxel Terrain by a Unimodal Terrain

Danny Z. Chen¹, *, Jinhee Chun², Naoki Katoh³, and Takeshi Tokuyama²

¹ Dept. of Computer Science and Engineering,
University of Notre Dame, Notre Dame, IN 46556, USA
`chen@cse.nd.edu`

² Graduate School of Information Sciences, Tohoku University, Sendai, Japan
`{jinhee,tokuyama}@dais.is.tohoku.ac.jp`

³ Graduate School of Engineering, Kyoto University, Kyoto, Japan
`naoki@archi.kyoto-u.ac.jp`

Abstract. We consider the problem of approximating a function on a d -dimensional voxel grid by a unimodal function to minimize the L_2 approximation error with respect to a given measure distribution on the grid. The output unimodal function gives a layered structure on the voxel grid, and we give efficient algorithms for computing the optimal approximation under a reasonable assumption on the shape of each horizontal layer. Our main technique is a dominating cut algorithm for a graph.

1 Introduction

Given a geometric object, transforming it into an object satisfying a certain geometric property is a problem frequently discussed in computational geometry. In this paper, we consider the following problem: Consider a function f in d variables as an input. We want to transform the function into a unimodal function $f_{\mathcal{P}}(x)$ minimizing the L_2 error (with respect to a given measure distribution) under the condition that each horizontal slice $\{x : f_{\mathcal{P}}(x) \leq t\}$ at height t has a good geometric shape. The computational complexity of the problem depends on the representation of input and output functions. For simplicity, we assume that the functions are defined on a voxel grid and investigate the complexity based on the number of voxels, although our framework works if f is defined on other types of subdivisions of \mathbb{R}^d such as triangulations by giving suitable modification.

Precisely speaking, we consider a d -dimensional voxel grid Γ consisting of $n = m_1 \times m_2 \times \dots \times m_d$ cells (voxels). We can assume $m_i = m$ for each $1 \leq i \leq d$ without loss of generality. We consider a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ that is a constant function in each voxel c . We can assume that f is nonnegative without loss of generality. A *grid region* is a simply connected region represented as a union of voxels. We fix a family \mathcal{R} of grid regions in Γ . Without loss of generality, we assume that $\emptyset \in \mathcal{R}$ and $\Gamma \in \mathcal{R}$.

* This author was supported in part by the National Science Foundation under Grant CCR-9988468.

Definition 1. A pyramid is a series \mathcal{P} of regions $P(t_i)$ ($i = 0, 1, 2, \dots, h$) in \mathcal{R} associated with a series of increasing nonnegative numbers (called heights) $t_0 < t_1 < t_2 < \dots < t_h$ satisfying that $P(t_0) = \Gamma$ and the monotone property $P(t) \subseteq P(t')$ for $t > t'$.

The surface function $f_{\mathcal{P}}$ of the pyramid P is defined by $f_{\mathcal{P}}(x) = t_i$ if $x \in P(t_i) - P(t_{i+1})$; or equivalently, $f_{\mathcal{P}}(x) = \max\{t_i : x \in P(t_i)\}$. Because of the monotone property of the pyramid and simply-connectedness of regions in \mathcal{R} , $f_{\mathcal{P}}$ is a unimodal function.

Given a measure function $\mu : \Gamma \rightarrow \mathbb{R}_{\geq 0}$, the squared L_2 distance between $f_{\mathcal{P}}$ and f is $(\|f - f_{\mathcal{P}}\|_2)^2 = \sum_{c \in \Gamma} \{\mu(c) \int_{x \in c} (f(x) - f_{\mathcal{P}}(x))^2 dx\}$. Since f is a constant in each voxel c , we can consider f as a function from Γ to \mathbb{R} by defining $f(c) = f(x_c)$ where x_c is any representative point in each voxel c . Similarly, we define $f_{\mathcal{P}}(c)$, and thus $(\|f - f_{\mathcal{P}}\|_2)^2 = \sum_{c \in \Gamma} \{\mu(c)(f(c) - f_{\mathcal{P}}(c))^2\}$. We can further rewrite it as $(\|f - f_{\mathcal{P}}\|_2)^2 = E(\mathcal{P}) = \sum_{i=0}^h \sum_{c \in P(t_i) \setminus P(t_{i+1})} (f(c) - t_i)^2 \mu(c)$. The optimal pyramid approximating f is the pyramid (identified to its surface function) minimizing this distance.

Although we consider a general measure function μ since it is essential in some applications, the problem is very intuitive if μ is the Euclidean volume function. See Figure 1 for an example where $d = 1$. The 1-dimensional problem is considered in [4], and a linear time solution has been given. If $d = 2$, the optimal pyramid (Fig 2) can be considered as reshaping a terrain defined by f into a single-peak terrain defined by $f_{\mathcal{P}}$ by moving earth from higher position to lower position minimizing the loss of positional potential (it will be shown in Lemma 1 that the total mass of the terrain is invariant). This seems to be a basic problem in computational geometry and geography, and some results are given by Chun *et al.* in [5].

Constructing an optimal pyramid in two or higher dimensions is a natural extension of the problem of region segmentation [2], and is useful in several applications in data mining, statistics, data visualization [1] geomorphology, and computer vision [3].

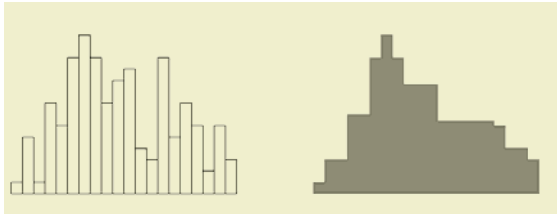


Fig. 1. A pyramid approximation (right) of a piecewise constant function f (left) for $d = 1$

In this paper, we investigate the condition under which the optimal pyramid can be efficiently constructed. Our condition is defined by using dominating sets of a directed graph. We can flexibly control the size of the family of regions by using the graph. In particular, we define the following d -dimensional region

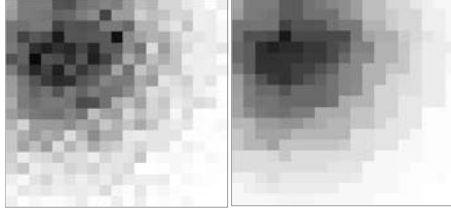


Fig. 2. Pyramid approximation (right) of a 2-dimensional terrain (left), where μ is constant and $f(c)$ and $f_P(c)$ are represented by using the gray levels of pictures (black and white represent 1 and 0, respectively)

families: (1) stabbed-union of orthogonal regions, (2) generalized base-monotone regions, and (3) digitized star-shaped regions. Surprisingly, for each of these region families, the optimal pyramid can be computed in polynomial time in the number n of voxels by reducing the problem to the minimum s - t cut problem in a directed graph.

Readers may worry that the number n of voxels may become large especially if d is high (say, more than three). However, we may combine voxels to obtain a compact structure to represent the functions to speed-up our algorithm.

2 Properties of the Optimal Pyramid

For a grid region R in Γ , we define $\rho(R) = \sum_{c \in R} f(c)\mu(c)$ and $\mu(R) = \sum_{c \in R} \mu(c)$. If it will cause no confusion, we also write $f(R)$ for $\rho(R)/\mu(R)$, which is weighted average of f over R .

Lemma 1. *For the optimal pyramid, $\sum_{c \in \Gamma} (f(c) - f_P(c))\mu(c) = 0$.*

Proof. The formula is rewritten as $\sum_{i=1}^h \rho(P(t_i) \setminus P(t_{i+1})) - t_i \cdot \mu(P(t_i) \setminus P(t_{i+1})) = 0$. It is well-known in statistics that in order to minimize $E(\mathcal{P}) = \sum_{i=0}^h \sum_{c \in P(t_i) \setminus P(t_{i+1})} (f(c) - t_i)^2 \mu(c)$, t_i must be the weighted average of $f(c)$ over $P(t_i) \setminus P(t_{i+1})$. This means that $t_i = \rho(P(t_i) \setminus P(t_{i+1}))/\mu(P(t_i) \setminus P(t_{i+1}))$, and we have the lemma.

We can consider that the input is the pair of ρ and μ , instead of μ and f . Introducing a nonnegative parameter t , $g_t(R, \rho, \mu) = \rho(R) - t \cdot \mu(R)$ is called the *parametric gain* of ρ against μ within the region R .

Lemma 2. *Consider a function $P(t)$ from $(0, \infty)$ to \mathcal{R} satisfying that $P(t) \subseteq P(t')$ for $t > t'$ and maximizing the objective function*

$$J(\mathcal{P}) = \int_{t=0}^{\infty} \rho(P(t)) - t \cdot \mu(P(t)) dt.$$

Then, it has at most $n + 1$ transition values $0 = t_{-1} < t_0 < t_1 < \dots < t_h$ satisfying that $P(t) = P(t_i)$ for $t \in (t_{i-1}, t_i]$ and $P(t) = \emptyset$ for $t > t_h$. Moreover, \mathcal{P} consisting of $P(t_0), P(t_1), \dots, P(t_h)$ is the optimal pyramid.

Proof. Since there are $n = m^d$ voxels in Γ , there are at most $n + 1$ different regions in $P(t)$ for $t \in (0, \infty)$. Thus, it is obvious that $P(t)$ has at most $n + 1$ transition values. Using Lemma 1, $J(\mathcal{P}) = \int_{t=0}^{\infty} \rho(P(t)) - t \cdot \mu(P(t)) dt = \sum_{i=0}^h \int_{t=t_{i-1}}^{t_i} \rho(P(t_i)) - t \cdot \mu(P(t_i)) dt = \sum_{i=0}^h \{(t_i - t_{i-1}) \cdot \rho(P(t_i)) - \frac{1}{2}(t_i^2 - t_{i-1}^2) \cdot \mu(P(t_i))\} = \sum_{i=0}^h t_i \cdot \rho(P(t_i) \setminus P(t_{i+1})) - t_i^2/2 \cdot \mu(P(t_i) \setminus P(t_{i+1})) = \sum_{i=0}^h t_i^2/2 \cdot \mu(P(t_i) \setminus P(t_{i+1}))$. On the other hand, using $t_i = \rho(P(t_i) \setminus P(t_{i+1}))/\mu(P(t_i) \setminus P(t_{i+1}))$, we have $E(\mathcal{P}) := \sum_{i=0}^h \sum_{c \in P(t_i) - P(t_{i+1})} (f(c) - t_i)^2 \cdot \mu(c) = \sum_{c \in \Gamma} \rho^2(c)/\mu(c) + \sum_{i=0}^h \{-2t_i \cdot \rho(P(t_i) - P(t_{i+1})) + t_i^2 \cdot \mu(P(t_i) - P(t_{i+1}))\} = \sum_{c \in \Gamma} \rho^2(c)/\mu(c) - \sum_{i=0}^h t_i^2 \cdot \mu(P(t_i) - P(t_{i+1}))$. Thus, $E(\mathcal{P}) = \sum_{c \in \Gamma} \rho^2(c)/\mu(c) - 2J(\mathcal{P})$. Obviously, $\sum_{c \in \Gamma} \rho^2(c)/\mu(c)$ is independent of the choice of the pyramid \mathcal{P} , and hence the function $P(t)$ maximizing $J(\mathcal{P})$ defines the pyramid \mathcal{P} minimizing $E(\mathcal{P})$. This proves the lemma.

Observing $J(\mathcal{P})$, intuitively, the optimal pyramid \mathcal{P} is obtained by piling up horizontal sections $P(t)$ with as large parametric gains as possible. Consider the region $R^{opt}(t)$ in \mathcal{R} maximizing the parametric gain $g_t(R, \rho, \mu)$. If t increases, $R^{opt}(t)$ tends to shrink. However, unfortunately, if we pile up the maximum gain regions $R^{opt}(t)$, they do not always form a pyramid, since $R^{opt}(t) \subset R^{opt}(t')$ does not always hold for $t > t'$.

2.1 Algorithms for a General Family

For $d = 1$, it is natural to consider the set of all integral intervals as \mathcal{R} . It has been shown that the optimal pyramid for the one-dimensional problem can be computed in $O(n)$ time [4].

In a higher dimensional case ($d \geq 2$), the time complexity largely depends on the specific family \mathcal{R} of regions. Indeed, it is not difficult to see that the problem of computing just a single flat of the optimal pyramid is NP-hard for some families even for $d = 2$ [2]. If \mathcal{R} has M different regions, the optimal pyramid can always be computed in polynomial time in M and n for the d -dimensional case. We construct a directed acyclic graph $G = (\mathcal{R}, E)$ whose vertex set is \mathcal{R} . For each pair R and R' of \mathcal{R} , we give a directed edge $e = (R, R')$ if and only if $R \supset R'$. We compute $t(e)$ such that $\rho(R \setminus R') = t(e) \cdot \mu(R \setminus R')$. The value $t(e)$ is called the *height label* of e , and $r(e) = t^2(e) \cdot \rho(R \setminus R')/2$ is called the *profit* of e . A directed path $p = e_0, e_1, \dots, e_q$ is called *admissible* if $t(e_{i-1}) < t(e_i)$ for $i = 1, 2, \dots, q$. The profit of an admissible directed path is the sum of profit values of the edges on it.

Lemma 3. *The optimal pyramid is associated with the admissible path with the maximum profit in G , such that $R \setminus R'$ is a flat of the pyramid if and only if (R, R') is an edge on that path.*

Thus, we can reduce the optimal pyramid problem to a maximum-weight-path problem in the directed acyclic graph G . Note that each directed path in G has at most n edges. By using a dynamic programming algorithm, we obtain the following result:

Theorem 1. *The optimal pyramid for \mathcal{R} of M different regions can be computed in $O(M^2n)$ time.*

Unfortunately, the above algorithm is seldom practical. For example, the family of rectangular regions has $O(n^2)$ regions, and hence the above time complexity is $O(n^5)$. Moreover, for computing accurate layered region rules, we want to consider families in which M is exponentially large in n . Thus, we seek to develop more efficient algorithms for some special families of regions.

3 Closed Family and Domination Closure of a Graph

A discrete family \mathcal{R} of regions in the d -dimensional space is called a *closed family* (under Boolean operations) if it is closed under the intersection and union operations, that is, $R \cap R' \in \mathcal{R}$ and $R \cup R' \in \mathcal{R}$ for any two regions R, R' in \mathcal{R} . The following proposition is easy to see:

Proposition 1. *Given a closed family \mathcal{R} , let $R^{opt}(t)$ be the region in \mathcal{R} maximizing $g_t(R, \rho, \mu)$. If there are multiple regions in \mathcal{R} maximizing $g_t(R, \rho, \mu)$, we take any one which is minimal under inclusion. Then, the series of transitions of $R^{opt}(t)$ gives the optimal pyramid \mathcal{P} for \mathcal{R} .*

We consider families \mathcal{R} that can be constructed using closed families as their building blocks.

3.1 Domination-Closures in a Grid Graph

In the voxel grid Γ , we first guess the cell $c = (c_1, c_2, \dots, c_d)$ giving the peak of the pyramid, and define a directed graph $G(c)$, whose vertex set consists of all voxels in Γ . For voxels $p = (p_1, p_2, \dots, p_d)$ and $q = (q_1, q_2, \dots, q_d)$, the L_1 distance between p and q is $dist(p, q) = \sum_{i=1}^d |p_i - q_i|$. The neighboring cells of p are the cells whose distance from p is one. For a cell p and its neighbor q , a directed edge is defined: its direction is (p, q) (i.e., from p to q) if $dist(p, c) = dist(q, c) + 1$ and otherwise (q, p) . The graph $G(c)$ is a weakly-connected directed graph with $d(m-1) \times m^{d-1} = O(n)$ edges (we assume d is a constant), and c is its unique sink vertex (i.e., a vertex without outgoing edges).

A subgraph $H = (V, E)$ of $G(c)$ is called a *rooted subgraph* if there exists a directed path in H from each vertex v of V to c .

Given a rooted subgraph $H = (V, E)$ of $G(c)$, we say that a vertex u is *H-dominated* by another vertex v if there exists a directed path from v to u in H . An *H-domination closure* W is a subset of V satisfying the condition that every H -dominated vertex by a vertex in W is contained in W . Clearly, each H -domination closure defines a grid region containing the cell c in Γ , and we like to identify such regions. Given a rooted subgraph H of $G(c)$, we consider the family \mathcal{R}_H that is the set of all H -domination closures. Since the domination closure property is closed under union and intersection, we have the following proposition:

Proposition 2. *For a rooted subgraph H of $G(c)$, \mathcal{R}_H is a closed family of regions.*

3.2 Algorithms for Computing the Optimal Pyramid with Respect to \mathcal{R}_H

We assume that $\mu(c)$ and $\rho(c)$ are quotient numbers of nonnegative integers less than U for every cell $c \in \Gamma$. In other words, U is the precision and $N = nU$ gives an upper bound of the bit size of the input.

Let us fix a rooted subgraph H of $G(c)$. We consider a parameter value t defining the height of a flat of the optimal pyramid with respect to \mathcal{R}_H , and we give a weight $\rho(p) - t \cdot \mu(p)$ to each voxel p (and its corresponding vertex in H). Due to the following lemma (proof is routine), we can assume that t is a rational number with small denominator and numerator.

Lemma 4. *If t is a height defining a flat of the optimal pyramid for a certain region family, then t is a rational number represented by a quotient of two integers less than or equal to N^2 .*

By definition, the maximum (parametric) gain region $P^{opt}(t) \in \mathcal{R}_H$ is the H -domination closure maximizing the sum of weights of voxels in the region. In graph-theoretic terminology, this is the *maximum domination closure* of the weighted directed graph H , and is obtained as the connected component of H containing c by removing a set of edges (the *cut set*). See Fig. 3 for a region $P^{opt}(t)$ in \mathcal{R}_H (here, $H = G(c)$ and $d = 2$) and the corresponding domination closure. The number in each pixel p of the left picture is the weight $\rho(p) - t \cdot \mu(p)$. The following theorem is due to Hochbaum [7] (also see [9]):

Theorem 2 (Hochbaum [7]). *Given a directed graph G with n vertices having real-valued vertex weights and m edges, its maximum domination closure can be computed in $O(T(n, m))$ time, where $T(n, m)$ is the time for computing a minimum s - t cut in an n -vertex, m -edge directed graph with nonnegative edge weights.*

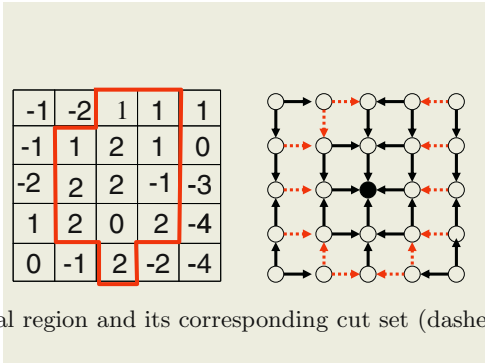


Fig. 3. An optimal region and its corresponding cut set (dashed arrows) in $G(c)$

Theorem 3. *The optimal pyramid for the family \mathcal{R}_H can be computed in $O(n^{1.5} \log n \log^2 N)$ time. The time complexity is improved to $O(n \log N)$ if H is a tree.*

Proof. We apply binary search for finding all possible heights of flats in the set S of quotient numbers defined by all pairs of integers less than or equal to N . We call the process “binary branching search” since we find more than one height. In the binary branching search, suppose we have $P^{opt}(t_0)$ and $P^{opt}(t_1)$ for $t_0 < t_1$. If $P^{opt}(t_0) = P^{opt}(t_1)$, we set the interval $I = (t_0, t_1)$ as *inactive*; otherwise, it is active and there may be a flat whose height is within the interval. If the interval is active, let t_{01} be the center (precisely, an element of S approximating the center) of (t_0, t_1) . Because the family \mathcal{R}_H is closed, $P^{opt}(t_1) \subset P^{opt}(t_{01}) \subset P^{opt}(t_0)$. Thus, to compute $P^{opt}(t_{01})$, we can remove all vertices outside $P^{opt}(t_0)$ from H and contract the vertices within $P^{opt}(t_1)$ into a single vertex. Thus, it suffices to compute the maximum domination closure in a directed graph $H(I)$ with $|P^{opt}(t_0)| - |P^{opt}(t_1)|$ vertices and the same order of edges (since the outdegree of each vertex remains to be at most two after the contraction).

The binary branching search consists of multiple levels, such that in a level, an active interval created at the previous level is split into two intervals. It is easy to see that the number of levels is $O(\log N)$. In each level, the sum of the numbers of vertices of $H(I)$ for all active intervals I is $O(n)$. We plug in $T(n, n) = O(n^{1.5} \log n \log N)$ [6], and hence the computation time for processing each level is $O(n^{1.5} \log n \log N)$. Therefore, the total time complexity is $O(n^{1.5} \log n \log^2 N)$. The improved time complexity for a tree is easy to obtain.

The algorithm given in Theorem 3 assumes that we know the peak c of the optimal pyramid, since the graph H depends on c . If we do not know the peak, a naive method is to examine all possible n cells of Γ as the candidates of the peak of the optimal pyramid, and report the one optimizing the objective function. This method takes $O(n^{2.5} \log n \log^2 N)$ time. However, we can practically guess the peak by to reduce the number of candidates of the peak to $n^\delta \ll n$, assuming that f has a good pyramid approximation. We omit describing the method in this version. If the method works, we can find the optimal pyramid in $O(n^{1.5+\delta} \log n \log^2 N)$ computation time without knowing the position of the peak in advance.

4 Algorithms for Typical Region Families

4.1 Stabbed Unions of Orthogonal Regions

For a fixed cell c of Γ , a region R in Γ is called a *stabbed union* of orthogonal regions at c if R can be represented as the union of orthogonal regions each of which contains c . The cell c is called the center cell of R .

Fig. 4 gives an example of the two-dimensional case. The pyramid given in Fig. 2 is based on the family of stabbed unions at a point. Naturally, the center cell (or point) gives the peak of the pyramid.

The following lemma is straightforward from the definitions of $G(c)$ and \mathcal{R}_H .

Lemma 5. *A region R in Γ is a stabbed union at c if and only if it is a $G(c)$ -domination closure.*

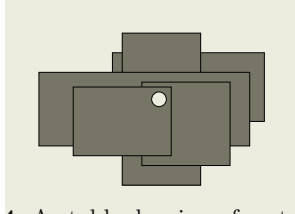


Fig. 4. A stabbed union of rectangles

Corollary 1. *The optimal pyramid for the family of stabbed unions of orthogonal regions at a given cell c can be computed in $O(n^{1.5} \log n \log^2 N)$ time.*

4.2 Based Monotone Regions

A based monotone region is a grid region R in Γ such that the intersection of R with each column of Γ is either empty or a continuous subcolumn whose lowest edge lies on the lower boundary of Γ ; That is, it is the connected lower half region given by a grid curve in Γ . The pyramid construction problem for the based monotone has been considered in [5].

The family of higher-dimensional based monotone regions containing a given fixed cell c is defined as follows. We fix a cell $c \in \Gamma$ in the d -dimensional voxel grid, and consider the graph $G(c)$ (defined in Section 3.1). For each vertex $p = (p_1, p_2, \dots, p_d) \neq c$, we find the largest index i such that $p_i \neq c_i$, and select the outgoing edge of p corresponding to its i -th coordinate. Thus, we obtain a spanning tree $T_0(c)$ of $G(c)$, which we call *lexicographic minimum bend* spanning tree. In the voxel grid, the path from p to c in $T_0(c)$ consists of at most d segments.

It is easy to see that a region R is a based monotone region containing $c = (c_1, 1)$ in a two-dimensional pixel grid if and only if it is a domination closure in $T_0(c)$. This gives a d -dimensional analogue of the family of based monotone regions. The following theorem is obvious from Theorem 3.

Theorem 4. *The optimal pyramid with respect to the family of domination closures of $T_0(c)$ can be computed in $O(n \log N)$ time.*

4.3 Digitized Star-Shaped Regions

In Euclidean geometry spaces (i.e., not the voxel grid settings), a popular closed family of regions is the family of star-shaped regions centered at a given point q . Recall that a region R is said to be *star-shaped* centered at q if for any point $v \in R$, the line segment \overline{qv} is entirely contained in R . The family of star-shaped regions is the closure with respect to the union and intersection of the family of convex regions containing q .

However, its digital analogue is not a well-defined problem in general, since it depends on the definition of the digital line segment between two voxels. Unfortunately, it does not form a closed family if we adopt one of the popular definitions

of digital line segments, and analogously define the star-shape polygons in the digital geometry.

Here, we give a probabilistic construction of the digital star-shaped regions by using $G(c)$. Let c be the center of the digital star-shaped regions. The first idea is to construct a spanning tree T of $G(c)$ rooted at c such that for each vertex v (i.e., a voxel of Γ) of T , the unique directed path $path_T(v, c)$ in T from v to c simulates a (true) line segment \overline{vc} in the digital space.

Each vertex of $G(c)$ has at most d outgoing edges. We obtain a spanning tree of $G(c)$ by selecting exactly one outgoing edge for each vertex (except c). Consider a vertex corresponding to the voxel $p = (p_1, p_2, \dots, p_d)$. Let $dist(p) = \sum_{i=1}^d |p_i - c_i|$ be the L_1 distance from p to c . Note that $dist(p)$ is also the distance from p to c in T . Our strategy for constructing T is to select the edge corresponding to the i -th coordinate with the probability $|p_i - c_i|/dist(p)$.

Let $x_i(k, p)$ be the number of edges corresponding to the i -th coordinate encountered during walking on the path $path_T(p, c)$ from p by k steps. Since T is constructed in a randomized fashion, $x_i(k, p)$ is a random variable.

Lemma 6. *The expectation $E = E(x_i(k, p))$ is $k|p_i - c_i|/dist(p)$. Moreover, the probability that $|x_i(k, p) - E| \geq \sqrt{2ak}$ is smaller than $2e^{-a}$ for any positive number a , where e is the natural logarithm base.*

Proof. The process is Martingale, and the expectation can be obtained by an induction on k . The deviation bound comes from Azuma's inequality [8].

This implies that $path_T(p, c)$ approximates the line segment \overline{pc} for each vertex p . However, the set of T -domination closures thus defined has a serious defect in simulating the star-shaped regions, even in the two dimensional case: Geometrically, a T -domination closure is not always a simply connected region in Γ . This is because there are some leaf vertices in T that do not correspond to any boundary cells (i.e., cells touching the grid boundary) of Γ .

For this purpose, for any leaf vertex v of T corresponding to an internal cell of Γ , we need to add an incoming edge to v in order to obtain a new graph \tilde{T} . Again, we choose one of the incoming edges for v in a probabilistic fashion. A simple calculation shows that the expected number of additional edges needed is $n/6 + o(n)$ if $d = 2$ and bounded by $n/3$ for $d \geq 3$. We then have a graph \tilde{T} as constructed above, and treat the set of domination closures in the graph \tilde{T} as a family of *digitized star-shaped regions* with the center c . See Fig. 5 for an example, and compare it with Fig. 3.

Corollary 2. *The optimal pyramid corresponding to \tilde{T} can be computed in $O(n^{1.5} \log^2 N)$ time.*

4.4 Reduction of the Voxel Size

Although our algorithms are polynomial in the voxel size n , n may become large and it is worried that the algorithm is too expensive to use in practice. For example, if $m = 100$ and $d = 4$, $n = 100,000,000$. In such a case, we need to

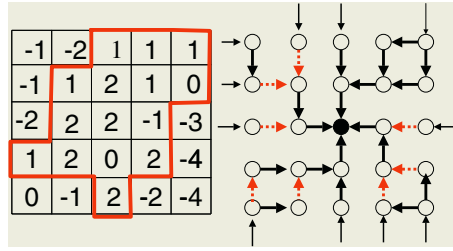


Fig. 5. An optimal region and its corresponding cut set in \tilde{T} (only a part near c is displayed)

preprocess the original input functions f and μ to reduce the input size for the algorithm; precisely speaking, to reduce the size of the graph $G(c)$ considered in the algorithm. For the purpose, we unite neighbor voxels with similar feature to make a large cell, and contract the corresponding vertices of $G(c)$ into one vertex to reduce the size of the graph. A typical method is the quad-tree method, in which we stop refining an intermediate cell if each voxel c in the cell has the values $f(c)$ and $\mu(c)$ that is near (i.e., difference is below a fixed threshold) to the average of those values over the cell. Although this does not improve the worst-case time complexity, we can replace n in the time complexities by n' that is the size of the reduced representation of the inputs by spending $O(n)$ preprocessing time.

5 Concluding Remarks

One drawback of our method is that the pixel structure is essential, and the algorithms does not work for a terrain given by using a polyhedral representation as it is. However, we can modify our algorithm to work for a locally constant function on a triangulation if we define a suitable directed graph (based on the dual graph of the triangulation) corresponding to the grid graph such that each domination-closure of the graph has nice geometric shape. A more general problem is approximation by a terrain with k peaks instead of a unimodal terrain; in other word, it gives a simplification of a Morse complex of a given input terrain. This general problem is a challenging problem.

References

1. A. Amir, R. Kashi, N. S. Netanyahu, Analyzing Quantitative Databases: Image Is Everything, *27th Proc. VLDB Conference* (2001).
2. T. Asano, D. Chen, N. Katoh, and T. Tokuyama, Efficient Algorithms for Optimization-Based Image Segmentation, *Int'l J. of Computational Geometry and Applications* **11**(2001) 145-166.

3. I. Bloch, Unifying Quantative, Semi-quantitive and Qualitative Spatial Relation Knowledge Representations Using Mathematical Morphology, *Proc. 11th Workshop on Theoretical Foundations of Computer Vision: Geometry, Morphology and Computational Imaging, LNCS 2616* (2003), pp.153-164.
4. J. Chun, K. Sadakane, T. Tokuyama, Linear Time Algorithm for Approximating a Curve by a Single-Peaked Curve, *Proc. 14th ISAAC, LNCS 2906* (2003), 6-16.
5. J. Chun, K. Sadakane, T. Tokuyama, Efficient Algoirthms for Constructing a Pyramid from a Terrain, *Proc. JCD CG2002, LNCS 2866* (2002), 108-117.
6. A. Goldberg and S. Rao, Beyond the Flow Decomposition Barrier, *Proc. 38th IEEE FOCS* (1997) 2–11.
7. D. S. Hochbaum, A New-old Algorithm for Minimum Cuts in Closure Graphs, *Networks* **37** (2001) 171-193.
8. R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge U. Press, 1995.
9. X. Wu and D. Z. Chen, Optimal Net Surface Problems with Applications, *Proc. 29th International Colloquium on Automata, Languages and Programming* (2002) 1029-1042.

Algorithms for Point Set Matching with k -Differences

Tatsuya Akutsu

Bioinformatics Center, Institute for Chemical Research, Kyoto University
Uji-city, Kyoto 611-0011, Japan
takutsu@kuicr.kyoto-u.ac.jp

Abstract. The largest common point set problem (LCP) is, given two point set P and Q in d -dimensional Euclidean space, to find a subset of P with the maximum cardinality that is congruent to some subset of Q . We consider a special case of LCP in which the size of the largest common point set is at least $(|P| + |Q| - k)/2$. We develop efficient algorithms for this special case of LCP and a related problem. In particular, we present an $O(k^3 n^{1.34} + kn^2 \log n)$ time algorithm for LCP in two dimensions, which is much better for small k than an existing $O(n^{3.2} \log n)$ time algorithm, where $n = \max\{|P|, |Q|\}$.

1 Introduction

Point set matching is one of well-studied problems in computational geometry and pattern recognition. Though approximate matching (in the sense of allowing errors in distances) is more important [4, 7, 11], several studies have been done on exact matching. The *congruence problem*, which asks whether two input point sets P and Q ($|P| = |Q| = n$) in d dimensions are congruent, is one of the fundamental exact matching problems. This problem can be solved in $O(n \log n)$ time in two [10, 13] and three [6] dimensions. For higher dimensions, Alt *et al.* developed an $O(n^{d-2} \log n)$ time algorithm [4], which was improved to randomized $O(n^{(d-1)/2} \log n)$ time by Akutsu [2], and was further improved to randomized $O(n^{d/4+O(1)})$ time by Matoušek (private communication, see also [2]).

The *largest common point set problem* (LCP) is also a fundamental exact matching problem [3]. This is a problem of, given two point sets P and Q in d -dimensional Euclidean space where $|P| = m$, $|Q| = n$ and $m \leq n$, finding a subset of P with the maximum cardinality that is congruent to some subset of Q . Akutsu, Tamaki and Tokuyama developed an $O((\lambda(n, m) + n^2) \log n)$ time algorithm in two dimensions, where $\lambda(n, m)$ is a combinatorial number denoting the inner product of distance-multiplicity vectors of two point sets [3]. They gave a non-trivial upper bound of $O(m^{1.77} n^{1.43})$ for $\lambda(n, m)$. They also gave algorithms in higher dimensions.

It is worthy to notice a large gap of time complexity between the congruence problem and LCP: $O(n \log n)$ time vs. $O((m^{1.77} n^{1.43} + n^2) \log n)$ time. Therefore, it is reasonable to ask whether this gap can be lessened if P and Q are very similar

Table 1. Comparison of time complexities for LCP with k -differences. It is assumed that the sizes of P , Q and the largest common point set are $\Theta(n)$, and $\log^{O(1)}(n)$ factors are ignored

	Dimension		
	1	2	3
this paper (deterministic)	$k^3 + n$	$k^3 n^{1.34} + kn^2$	$k^4 n^{1.8} + k^2 n^{2.5}$
[3] (deterministic)	n^3 (trivial)	$n^{3.2}$	$n^{4.69}$
this paper (randomized)	$k^2 + n$	$kn^{1.34} + n^2$	$kn^{1.8} + n^{2.5}$
[3] (randomized)	n^2 (trivial)	$n^{2.2}$	$n^{2.69}$

(i.e., the largest common point set is close to P and Q). Precisely, we consider a special case of LCP in which at most k points are deleted from $P \cup Q$ to obtain the largest common point set. In other words, we consider a special case of LCP in which the size of the largest common point set is at least $(m+n-k)/2$. We call this special case *LCP with k -differences*. We develop an $O(k^3 n^{4/3} + kn^2 \log n)$ time algorithm for this special case, which is much better for small k than the $O((m^{1.77} n^{1.43} + n^2) \log n)$ time algorithm. We also develop an $O(kn^{4/3} \log n + n^2 \log^2 n)$ time randomized algorithm which is also better for small k than a previous $O((m^{1.77} n^{1.43} + n^2) K^{-1} \log n)$ time randomized algorithm [3], where K denotes the size of the largest common point set. We also developed algorithms in one and three dimensions. The results on LCP are summarized in Table 1. These show that it is possible to lessen a complexity gap between the congruence problem and LCP if P and Q are very similar.

Though it is not so common to consider k -differences problems in point set matching, a lot of studies have been done for *string matching with k -differences* [9, 12]. In particular, Landau and Vishkin developed an $O(kn)$ time algorithm for string matching with k -differences [12], where n denotes the length of a text string. One of the main ideas in their algorithm is the use of a suffix tree for finding a maximally matching substring pair in constant time. We also employ this idea to identify a maximally matching segment pair in constant time, which allows us to break the barrier of $\lambda(n, m)$ factor in the time complexity. Though exact string matching was applied to the congruence problem in two dimensions [10, 13], simple replacement of exact string matching with the Landau-Vishkin algorithm does not lead to an algorithm for LCP.

In this paper, we show another and more practical application of the ideas in the Landau-Vishkin algorithm to a geometric problem. We consider the *k -differences problem for numerical sequences*. Many scientific data are represented as *line plot* graphs, where each line plot corresponds to a sequence of numerical data. For example, time series data of gene expression levels are considered to be a set of line plot graphs [18], where each line plot corresponds to time series of one gene. It is important to compare time series data because genes may have similar functions if their time series data are similar [18]. The k -differences problem for numerical sequences can be trivially solved in $O(mn)$ time, where m denotes the length of pattern sequence and n denotes the length of text sequence.

Combining the ideas in [1], [5] and [12], we develop an $O(k^{1/3}m^{2/3}n \log n)$ time algorithm, which is better than the $O(mn)$ time algorithm for small k .

In this paper, we adopt a random access machine (RAM) as a model of computation. As usual, we assume that the machine can store each $O(\log n)$ bit integer in a word and can perform each of arithmetic and logical operations in constant time. Furthermore, we assume that for handling coordinates of points, the machine can represent arbitrary real numbers and can exactly perform all the geometric computations involved (e.g., determining angles, distances, etc.) without round-off-errors. However, the latter can be weakened without increasing the orders of the time complexities so that each of addition, subtraction, multiplication and division on rational numbers can be performed in constant time with finite precision. Though we do not consider mirror images, all the results are valid even if rigid motions with mirror images are considered.

2 One-Dimensional Case

Before describing algorithms, we formally define LCP with k -differences (for any fixed dimension d). Let $P = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_m\}$ and $Q = \{\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n\}$ be point sets in d -dimensional Euclidean space, respectively. Let k be a positive integer. Then, the largest common point set problem with k -differences (which is abbreviated as LCP- k DIFF) is defined as follows.

LCP- k DIFF (Largest common point set with k -differences)

Given P , Q and k , find a rigid motion \mathcal{T} such that $|\mathcal{T}(P) \cap Q|$ is the maximum under the condition that $|\mathcal{T}(P) \cap Q| \geq (m + n - k)/2$ and output the size of $|\mathcal{T}(P) \cap Q|$. If such \mathcal{T} does not exist, output “not found”.

It should be noted that the number of differences (i.e., the number of points not contributing to the largest common point set) is k' if and only if $|\mathcal{T}(P) \cap Q|$ is $(m + n - k')/2$. We use LCP also to denote the largest common point set.

LCP in one dimension can be solved in $O(m^2n \log n)$ time by using the following naive algorithm. For each pair $(\mathbf{p}_{i_0}, \mathbf{q}_{j_0})$, we consider the uniquely determined rigid motion \mathcal{T} such that $\mathcal{T}(\mathbf{p}_{i_0}) = \mathbf{q}_{j_0}$. Then, we compute $\mathcal{T}(P) \cap Q$ by examining for each \mathbf{p}_i whether there exists \mathbf{q}_j such that $\mathcal{T}(\mathbf{p}_i) = \mathbf{q}_j$ using binary search. Clearly, $\mathcal{T}(P) \cap Q$ can be computed in $O(m \log n)$ time, from which the total $O(m^2n \log n)$ computation time follows.

Here we assume w.l.o.g. (without loss of generality) that P and Q are sorted in increasing order (i.e., $\mathbf{p}_1 < \mathbf{p}_2 < \dots < \mathbf{p}_m$, $\mathbf{q}_1 < \mathbf{q}_2 < \dots < \mathbf{q}_n$). For each pair $(\mathbf{p}_i, \mathbf{q}_j)$, we define $mmsp(\mathbf{p}_i, \mathbf{q}_j)$ (maximally matching segment pair) to be the pair of $(\mathbf{p}_i, \mathbf{p}_{i+1}, \mathbf{p}_{i+2}, \dots, \mathbf{p}_{i+h-1})$ and $(\mathbf{q}_j, \mathbf{q}_{j+1}, \mathbf{q}_{j+2}, \dots, \mathbf{q}_{j+h-1})$ with the maximum h such that $\mathbf{p}_{i+h'} - \mathbf{p}_i = \mathbf{q}_{j+h'} - \mathbf{q}_j$ holds for all $h' < h$ (see also Fig. 1). We denote such h by $|mmsp(\mathbf{p}_i, \mathbf{q}_j)|$.

Lemma 1. *There exists a data structure with which $|mmsp(\mathbf{p}_i, \mathbf{q}_j)|$ can be answered in $O(1)$ time for given (i, j) . Moreover, the data structure can be constructed in $O(n \log n)$ time.*

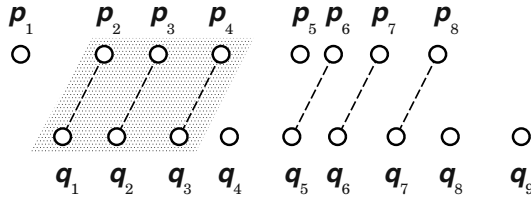


Fig. 1. Example of LCP- k DIFF in one dimension, where $k = 5$. Matching point pairs (i.e., points in LCP) are connected by dashed lines. Shaded region shows $mmisp(p_2, q_1)$

Proof. We use a technique developed in [12]. We define an alphabet Σ by

$$\Sigma = \{p_{i+1} - p_i | i = 1, \dots, m-1\} \cup \{q_{j+1} - q_j | j = 1, \dots, n-1\},$$

where identical vectors are regarded as the same element. We construct a *suffix tree* for string $str_{P,Q}$ defined by

$$str_{P,Q} = (p_2 - p_1, p_3 - p_2, \dots, p_m - p_{m-1}, \#, q_2 - q_1, q_3 - q_2, \dots, q_n - q_{n-1}, \$),$$

where $\#, \$ \notin \Sigma$. $O(n \log n)$ time is sufficient for construction of Σ and the suffix tree [9, 14]. Then, we can get $|mmisp(p_i, q_j)|$ in $O(1)$ time by computing LCA (lowest common ancestor) between the leaf corresponding to the suffix starting at $p_{i+1} - p_i$ and the leaf corresponding to the suffix starting at $q_{j+1} - q_j$, with $O(n)$ preprocessing time [15]. \square

The procedure $LCPK_{1D}(P, Q, k)$ outlined below computes the minimum difference k_{min} , which corresponds to LCP with k_{min} differences.

Procedure $LCPK_{1D}(P, Q, k)$

Construct a suffix tree for $str_{P,Q}$;

$k_{min} \leftarrow k + 1$;

for $i_0 = 1$ **to** $k + 1$ **do**

for $j_0 = 1$ **to** $k + 1$ **do**

$k' \leftarrow i_0 + j_0 - 2$; $i \leftarrow i_0$; $j \leftarrow j_0$;

 Let \mathcal{T} be a rigid motion such that $\mathcal{T}(p_{i_0}) = q_{j_0}$;

while $i \leq m$ **and** $j \leq n$ **and** $k' \leq k$ **do**

$h \leftarrow |mmisp(p_i, q_j)|$;

 Find the minimum i' such that $i' \geq i + h$ and $(\exists j')(\mathcal{T}(p_{i'}) = q_{j'})$;
 - (#1)

 (If such i' does not exist within total error k ,
 then exit while loop and update k' appropriately)

$k' \leftarrow k' + (i' - i - h) + (j' - j - h)$; $i \leftarrow i'$; $j \leftarrow j'$;

if $k' < k_{min}$ **then** $k_{min} \leftarrow k'$;

if $k_{min} \leq k$ **then** output k_{min} **else** output “not found”

This procedure is similar to the Landau-Vishkin algorithm [12]. However, we do not use dynamic programming because $p_{i+l} - p_i$ should be computed if

$\mathbf{p}_{i+1}, \dots, \mathbf{p}_{i+l-1}$ do not contribute to LCP . Instead, we can use the rigid motion (uniquely determined by $(\mathbf{p}_{i_0}, \mathbf{q}_{j_0})$) to locate the next matching pair.

Theorem 1. *LCP - $kDIFF$ in one dimension can be solved in $O(k^3 + n \log n)$ time.*

Proof. The correctness of $LCPK_{1D}(P, Q, k)$ is obvious because at most k points are deleted from $P \cup Q$ and we need to examine at most $(k+1)^2$ rigid motions.

Here we consider the time complexity. The while loop is repeated at most $k+1$ times per (i_0, j_0) . Within the while loop, $O(1)$ time is sufficient except for the part of (#1). (#1) is computed as in the merge phase of the merge sort (i.e., i is incremented if $\mathcal{T}(\mathbf{p}_i) < \mathbf{q}_j$, otherwise j is incremented). Since i and j are incremented at most $2k$ times in total, the total time required for (#1) is $O(k)$ per (i_0, j_0) . Therefore, $O(k)$ time is sufficient per (i_0, j_0) . Since $O(k^2)$ pairs of (i_0, j_0) are examined, the total time is $O(k^3 + n \log n)$. \square

In $LCPK_{1D}(P, Q, k)$, we examined $k+1$ points from P . However, if we randomly sample a constant number of points P' from P (and corresponding $O(k)$ points from Q), the probability that P' contains at least one point in LCP is high if k is not too large (e.g., $k < m/c$). The probability of error can be made small (e.g., less than $1/n^{c'}$) if we repeat the procedure $O(\log n)$ times.

Corollary 1. *Suppose that c is an arbitrary fixed positive constant and $k < m/c$ holds. Then, LCP - $kDIFF$ in one dimension can be solved in $O((k^2 + n) \log n)$ time with high probability.*

Though we considered a kind of global matching in the above, we can also consider local matching as in approximate string matching. In this case, we only count the errors (the number of points in $\mathcal{T}(P) \cup Q$ not included in LCP) in the range of $[\mathcal{T}(\mathbf{p}_1), \mathcal{T}(\mathbf{p}_m)]$. We call this problem LCP - $kDIFF$ -LOCAL.

Corollary 2. *LCP - $kDIFF$ -LOCAL in one dimension can be solved in $O(k^2n + n \log n)$ time deterministically, and in $O(kn \log n)$ time by a randomized algorithm with high probability assuming $k < m/c$.*

3 Two-Dimensional Case

To solve LCP - $kDIFF$ in two dimensions, we reduce the two-dimensional case to a problem similar to LCP - $kDIFF$ -LOCAL.

Suppose (until Lemma 2) that we have two point sets $P' = P \cup \{\mathbf{p}_0\}$ and $Q' = Q \cup \{\mathbf{q}_0\}$, where $\mathbf{p}_0 \notin P$ must matches $\mathbf{q}_0 \notin Q$ in a rigid motion. We sort the points of P in the following way. Let $\theta(\mathbf{p}_i)$ ($0 \leq \theta(\mathbf{p}_i) < 2\pi$) to be the angle of $\mathbf{p}_i - \mathbf{p}_0$ with respect to $\mathbf{p}_1 - \mathbf{p}_0$ (i.e., $\theta(\mathbf{p}_i) = \angle \mathbf{p}_i \mathbf{p}_0 \mathbf{p}_1$). Let $|\mathbf{x}|$ denote the length of a vector \mathbf{x} . Then, we define the total order \prec on P by

$$\mathbf{p}_i \prec \mathbf{p}_{i'} \iff \theta(\mathbf{p}_i) < \theta(\mathbf{p}_{i'}) \text{ or } (\theta(\mathbf{p}_i) = \theta(\mathbf{p}_{i'}) \text{ and } |\mathbf{p}_i - \mathbf{p}_{i_0}| < |\mathbf{p}_{i'} - \mathbf{p}_{i_0}|).$$

The points of Q are sorted in the same way (see also Fig. 2).

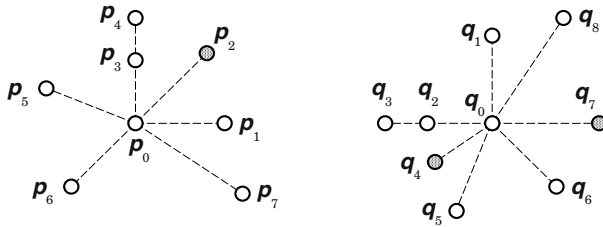


Fig. 2. Example of LCP- k DIFF in two dimensions. Points are sorted according to “ \prec ”. Points not included in LCP are shaded

Now we can assume w.l.o.g. that $p_1 \prec p_2 \prec \dots \prec p_m$, and $q_1 \prec q_2 \prec \dots \prec q_n$. We define an equivalence relation \sim on $\langle p_{i+1}, p_i \rangle$ and $\langle q_{j+1}, q_j \rangle$ by

$$\begin{aligned} \langle p_{i+1}, p_i \rangle \sim \langle q_{j+1}, q_j \rangle &\iff |p_{i+1} - p_0| = |q_{j+1} - q_0| \text{ and } |p_i - p_0| = |q_j - q_0| \\ &\text{and } \angle p_{i+1} p_0 p_i = \angle q_{j+1} q_0 q_j, \end{aligned}$$

where this relation is extended to $\langle p_{i+1}, p_i \rangle$'s (resp. $\langle q_{j+1}, q_j \rangle$'s). As in the one-dimensional case, we define an alphabet Σ' by $\Sigma' = \{\langle p_{i+1}, p_i \rangle | i = 1, \dots, m\} \cup \{\langle q_{i+1}, q_i \rangle | i = 1, \dots, n\}$, where $p_{m+1} \equiv p_1$, $q_{n+1} \equiv q_1$, and equivalent elements are regarded as the same element.

Here, we construct a sequence $PP = (p'_1, \dots, p'_{2m-1})$ where $p'_i = p_i$ if $i \leq m$, otherwise $p'_i = p_{i-m}$. We also construct a sequence $QQ = (q'_1, \dots, q'_{2n-1})$ in an analogous way. Then, we construct a suffix tree for $str_{PP,QQ}$ over Σ' defined by

$$\begin{aligned} str_{PP,QQ} = (&\langle p'_2, p'_1 \rangle, \langle p'_3, p'_2 \rangle, \dots, \langle p'_{2m-1}, p'_{2m-2} \rangle, \#, \\ &\langle q'_2, q'_1 \rangle, \langle q'_3, q'_2 \rangle, \dots, \langle q'_{2n-1}, q'_{2n-2} \rangle, \$). \end{aligned}$$

Clearly, all the constructions can be done in $O(n \log n)$ time.

For each rigid motion \mathcal{T} satisfying $\mathcal{T}(p_0) = q_0$ and $\mathcal{T}(p'_i) = q'_j$ where $i \leq k+1$, we compute a common point set $\mathcal{T}(\{p'_i, \dots, p'_{i+m-1}\}) \cap \{q'_j, \dots, q'_{j+n-1}\}$ as in the one-dimensional case. Since only k -differences are allowed, it takes $O(k)$ time per \mathcal{T} and thus $O(kn)$ time in total.

Lemma 2. *Let p_i and q_j are arbitrary points chosen from P and Q , respectively. Suppose that only rigid motions \mathcal{T} satisfying $\mathcal{T}(p_i) = q_j$ are allowed. Then, LCP- k DIFF in two dimensions can be solved in $O(k^2 n + n \log n)$ time.*

In order to solve LCP- k DIFF, it is sufficient to examine $O(kn)$ pairs of (p_0, q_0) . Precisely, we select $k+1$ points from P and all points from Q because at least one point in the selected $k+1$ points contributes to LCP . Therefore, it takes $O(k^3 n^2 + kn^2 \log n)$ time.

However, this time can be reduced using the fact that there exists $O(n^{4/3})$ point pairs which have the same distance in two dimensions [16, 17]. Therefore, at most $O(k^2 n^{4/3})$ rigid motions are examined and thus $O(k^3 n^{4/3})$ time is required in total along with $O(kn^2 \log n)$ time for pre-processing. Enumeration of such

motions can be done in $O(k^2 n^{4/3} \log n)$ time with $O(n^2 \log n)$ pre-processing time. Since $k^2 n^{4/3} \log n$ is not greater than $O(k^3 n^{4/3} + kn^2 \log n)$, we have:

Theorem 2. *LCP- k DIFF in two dimensions can be solved in $O(k^3 n^{4/3} + kn^2 \log n)$ time.*

We can also develop a randomized algorithm in two dimensions. In this case, we only sample a constant number of pairs $(\mathbf{p}_0, \mathbf{p}_i)$. If $k < m/c$ holds for arbitrary fixed constant c , it is expected that both points in the pair are included in LCP with no less than some constant probability. For each pair, we enumerate pairs $(\mathbf{q}_0, \mathbf{q}_j)$ satisfying $|\mathbf{p}_i - \mathbf{p}_0| = |\mathbf{q}_j - \mathbf{q}_0|$. Enumeration of such pairs can be done in $O(n^{4/3} \log n)$ time with $O(n^2 \log n)$ pre-processing time. Then, we examine each rigid motion \mathcal{T} defined by $\mathcal{T}(\mathbf{p}_0) = \mathbf{q}_0$ and $\mathcal{T}(\mathbf{p}_i) = \mathbf{q}_j$. Since $O(n^{4/3})$ rigid motions are examined and $O(n^2 \log n)$ time is required for constructing Σ 's and suffix trees in total, this randomized algorithm takes $O(kn^{4/3} + n^2 \log n)$ time. We repeat this algorithm $O(\log n)$ times in order to increase the success probability.

Corollary 3. *Suppose that c is an arbitrary fixed positive constant and $k < m/c$ holds. Then, LCP- k DIFF in two dimensions can be solved in $O(kn^{4/3} \log n + n^2 \log^2 n)$ time with high probability.*

4 Three-Dimensional Case

The algorithms for the two-dimensional case can be modified for three dimensions. We describe the required modifications in this section.

In order to reduce the three-dimensional problem to a string matching like problem, we use two point pairs $(\mathbf{p}_{i_1}, \mathbf{q}_{j_1})$ and $(\mathbf{p}_{i_2}, \mathbf{q}_{j_2})$ and consider rotations around $\overline{\mathbf{p}_{i_1}\mathbf{p}_{i_2}}$ and $\overline{\mathbf{q}_{j_1}\mathbf{q}_{j_2}}$ (see Fig. 3). We can define the total order \prec on P by using another point \mathbf{p}_{i_3} and considering the following for each $\mathbf{p}_i \in P$:

- the angle between the planes defined by $\mathbf{p}_{i_1}, \mathbf{p}_{i_2}, \mathbf{p}_i$ and by $\mathbf{p}_{i_1}, \mathbf{p}_{i_2}, \mathbf{p}_{i_3}$,
- $(\mathbf{p}_i - \mathbf{p}_{i_1}) \cdot (\mathbf{p}_{i_2} - \mathbf{p}_{i_1})$ where $\mathbf{x} \cdot \mathbf{y}$ denote the inner product,
- distance between \mathbf{p}_i and $\overline{\mathbf{p}_{i_1}\mathbf{p}_{i_2}}$.

The total order on Q is defined in an analogous way. Once the total orderings are defined, we can use the following procedure.

- (i) for $O(k^2)$ pairs of $(\mathbf{p}_{i_1}, \mathbf{p}_{i_2})$, for pairs of $(\mathbf{q}_{j_1}, \mathbf{q}_{j_2})$ such that $|\mathbf{p}_{i_2} - \mathbf{p}_{i_1}| = |\mathbf{q}_{j_2} - \mathbf{q}_{j_1}|$, execute (ii)-(iii),
- (ii) sort P, Q and construct a suffix tree,
- (iii) for $O(k)$ points of \mathbf{p}_{i_3} , for \mathbf{q}_{j_3} and \mathcal{T} satisfying $\mathcal{T}(\triangle \mathbf{p}_{i_1}\mathbf{p}_{i_2}\mathbf{p}_{i_3}) = \triangle \mathbf{q}_{j_1}\mathbf{q}_{j_2}\mathbf{q}_{j_3}$, compute $\mathcal{T}(P) \cap Q$ under the condition that at most k -differences are allowed.

Theorem 3. *LCP- k DIFF in three dimensions can be solved in $O(k^4 n^{1.8} \log n + k^2 n^{5/2} \log^2 n)$ time.*

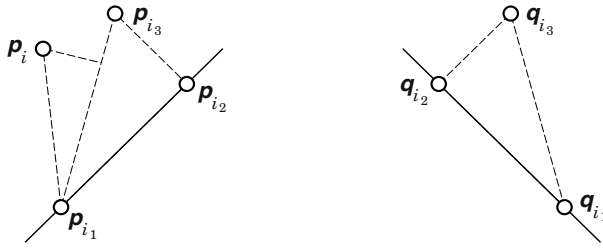


Fig. 3. LCP- k DIFF in three dimensions. For each rigid motion \mathcal{T} such that $\mathcal{T}(\triangle p_{i_1} p_{i_2} p_{i_3}) = \triangle q_{i_1} q_{i_2} q_{i_3}$, $\mathcal{T}(P) \cap Q$ is computed

Proof. Since the correctness is obvious, we consider the time complexity.

For each (p_{i_1}, p_{i_2}) , there are $O(n^{3/2} \log n)$ (precisely, $O(n^{3/2} (\log^* n)^{O(1)})$) pairs of (q_{j_1}, q_{j_2}) satisfying the required condition [8]. Since $O(n \log n)$ time is enough per execution, step (ii) takes $O(k^2 n^{5/2} \log^2 n)$ time in total.

The number of congruent triangles in three dimensions is $O(n^{1.8} \log n)$ (precisely, $O(n^{1.8} (\log^* n)^{O(1)})$) [3]. Thus, $O(k^3 n^{1.8} \log n)$ triangles are examined and step (iii) takes $O(k^4 n^{1.8} \log n + k^3 n^{1.8} \log^2 n)$ time in total. Since $k^3 n^{1.8} \log^2 n$ is no greater than $O(k^4 n^{1.8} \log n + k^2 n^{5/2} \log^2 n)$, the theorem holds. \square

Corollary 4. Suppose that c is an arbitrary fixed positive constant and $k < m/c$ holds. Then, LCP- k DIFF in three dimensions can be solved in $O(kn^{1.8} \log^2 n + n^{5/2} \log^3 n)$ time with high probability (using random sampling of $\triangle p_{i_1} p_{i_2} p_{i_3}$).

5 Approximate Matching of Numerical Sequences

Let $P = (p_1, \dots, p_m)$ and $Q = (q_1, \dots, q_n)$ be sequences of real numbers. We say that p_i matches q_j if $|p_i - q_j| < \epsilon$, where ϵ is a real number given as a part of an input. Then, the k -mismatches problem for numerical sequences is to enumerate all positions j of Q such that $|p_i - q_{j+i-1}| < \epsilon$ holds for at least $m - k$ indices i . The k -differences problem for numerical sequences is to enumerate all positions j such that $|p_{i_l} - q_{j+j_l-1}| < \epsilon$ holds for all $l = 1, \dots, h$, where $1 \leq i_1 < i_2 < \dots < i_h \leq m$ and $1 \leq j_1 < j_2 < \dots < j_h$ hold, and the number of errors (mismatches, deletions and insertions) $j_h + m - 2h$ is at most k (see also Fig. 4). It is easy to see that the k -differences problem (and also the k -mismatches problem) can be solved in $O(mn)$ time using dynamic programming. Here, we describe an $o(mn)$ time algorithm for small k .

The proposition below directly follows from the result on the smaller matching problem [5], which is to enumerate all positions j of Q such that $p_i < q_{j+i-1}$ holds for all $i = 1, \dots, m$.

Proposition 1. The k -mismatches problem for numerical sequences can be solved in $O(\sqrt{mn} \log m)$ time.

In order to develop an algorithm for the k -differences problem, we modify the Landau-Vishkin algorithm. One of the main ideas in Landau-Vishkin algorithm

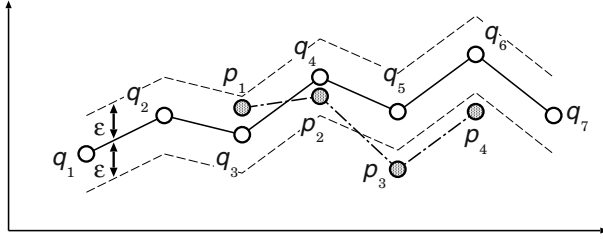


Fig. 4. Approximate matching of numerical sequences. In this case, we have a match $\{(p_1, q_3), (p_2, q_4), (p_4, q_5)\}$ with 1 difference (p_3 is considered to be deleted from P)

is fast computation of $|mmsp(p_i, q_j)|$, where $mmsp(p_i, q_j)$ is defined here to be a pair of $(p_i, p_{i+1}, \dots, p_{i+h-1})$ and $(q_j, q_{j+1}, \dots, q_{j+h-1})$ with the maximum h such that $p_{i+h'}$ matches $q_{j+h'}$ for all $h' < h$. If we could use a suffix tree, $|mmsp(p_i, q_j)|$ would be computed in $O(1)$ time. However, we can not use a suffix tree here because p_i can match multiple q_j 's (having non-identical values). Therefore, we employ a technique introduced in [1] so as to efficiently (but less efficiently than using suffix trees) compute $|mmsp(p_i, q_j)|$.

Let M ($M < m$) be an integer whose exact value is to be determined later. We divide P into m/M blocks $P_1, P_2, \dots, P_{m/M}$, where each P_i has length M and we assume w.l.o.g. that m/M is an integer. For each of P_t , we compute a table $W[t, j]$, where $W[t, j]$ denotes the maximum number of blocks s such that P_t, \dots, P_{t+s-1} matches $q_j, q_{j+1}, \dots, q_{j+Ms-1}$. Once this table is constructed, we can compute $|mmsp(p_i, q_j)|$ in $O(M)$ time: we check each pair (p_{i+l}, q_{j+l}) one by one until $i + l \bmod M = 1$, and then we look up $W[(i + l - 1/M) + 1, j + l]$ (let w be that value) and jump to the pair (p_{i+l+wM}, q_{j+l+wM}) , after which we need to check at most M pairs. Since $|mmsp(p_i, q_j)|$ is computed in $O(M)$ time, we can modify the Landau-Vishkin algorithm so that it works in $O(kMn)$ time.

Table $W[t, j]$ is constructed as follows. For each block P_i , we solve 1-mismatch problem for numerical sequences using Proposition 1 and enumerate all positions j of Q such that P_i matches $q_j, q_{j+1}, \dots, q_{j+M-1}$. $W[t, j]$ is initially set to 1 if P_i matches with $q_j, q_{j+1}, \dots, q_{j+M-1}$, otherwise $W[t, j]$ is set to 0. After this, $W[t, j]$ is replaced by the maximum number w such that $W[t, j] = W[t + 1, j + M] = \dots = W[t + w - 1, j + (w - 1)M]$. It is not difficult to see that table construction can be done in $O(\frac{mn}{M} \sqrt{M} \log m)$ time.

Solving $kMn = \frac{mn}{M} \sqrt{M}$, we have $M = (\frac{m}{k})^{2/3}$. In practice, it is enough to set M an integer nearest to $M = (\frac{m}{k})^{2/3}$.

Theorem 4. *The k -differences problem for numerical sequences can be solved in $O(k^{1/3} m^{2/3} n \log m)$ time.*

Acknowledgements

This work was supported in part by a Grant-in-Aid #16300092 from the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan.

References

1. Akutsu, T.: Approximate String Matching with Don't Care Characters. *Information Processing Letters* **55** (1995) 235–239
2. Akutsu, T.: On Determining the Congruence of Point Sets in d Dimensions. *Computational Geometry. Theory and Applications* **9** (1998) 247–256
3. Akutsu, T., Tamaki, H., Tokuyama, T.: Distribution of Distances and Triangles in a Point Set and Algorithms for Computing the Largest Common Point Sets. *Discrete and Computational Geometry* **20** (1998) 307–331
4. Alt, H., Melhorn, K., Wagener, H., Welzl, E.: Congruence, Similarity, and Symmetrics of Geometric Objects. *Discrete and Computational Geometry* **3** (1988) 237–256
5. Amir, A., Farach, M.: Efficient 2-Dimensional Approximate Matching of Half-Rectangular Figures. *Information and Computation* **118** (1995) 1–11
6. Atkinson, M.D.: An Optimal Algorithm for Geometrical Congruence. *J. Algorithms* **8** (1987) 159–172
7. Cardoze, D.E., Schulman, L.J.: Pattern Matching for Spatial Point Sets. *Proc. 38th Symp. Foundations of Computer Science* (1998) 156–165
8. Clarkson, K., Edelsbrunner, H., Guibas, L., Sharir, M., Welzl, E.: Combinatorial Complexity Bounds for Arrangements of Curves and Spheres. *Discrete and Computational Geometry* **5** (1990) 99–160
9. Crochemore, M., Rytter, W.: *Jewels of Stringology*. World Scientific, Singapore (2002)
10. Highnam, P.T.: Optimal Algorithms for Finding the Symmetries of a Planar Point Set. *Information Processing Letters* **18** (1986) 219–222
11. Indyk, P., Venketasubramanian, S.: Approximate Congruence in Nearly Linear Time. *Computational Geometry. Theory and Applications* **24** (2003) 115–128
12. Landau, G.M., Vishkin, U.: Fast Parallel and Serial Approximate String Matching. *J. Algorithms* **10** (1989) 157–169
13. Manacher, G.: An Application of Pattern Matching to a Problem in Geometrical Complexity. *Information Processing Letters* **5** (1976) 6–7
14. McCright, E.M.: A Space-Economical Suffix Tree Construction Algorithm. *J. ACM* **23** (1976) 262–272
15. Schieber, B., Vishkin, U.: On Finding Lowest Common Ancestors: Simplification and Parallelization. *SIAM J. Computing* **17** (1988) 1253–1262
16. Székely, L.: Crossing Numbers and Hard Erdős Problems in Discrete Geometry. *Combinatorics, Probability and Computing* **6** (1997) 353–358
17. Szemerédi, E., Trotter, W.T.: Extremal Problems in Discrete Geometry. *Combinatorica* **3** (1983) 381–392
18. Tamayo, P., Slonim, D., Mesirov, J., Zhu, Q., Kitareewan, A., Dmitrovsky, E., Lander, E.S., Golub, T.R.: Interpreting Patterns of Gene Expression with Self-Organizing Maps: Methods and Application to Hematopoietic Differentiation. *Proc. Natl. Acad. Sci. USA* **96** (1999) 2907–2912

Approximation Algorithms for Inscribing or Circumscribing an Axially Symmetric Polygon to a Convex Polygon^{*}

Hee-Kap Ahn¹, Peter Brass², Otfried Cheong³, Hyeon-Suk Na⁴,
Chan-Su Shin⁵, and Antoine Vigneron⁶

¹ Korea Advanced Institute of Science and Technology, Daejeon, Korea
`heekap@hanmail.net`

² Dept. of Computer Science, City College of New York, USA
`peter@cs.ccny.cuny.edu`

³ Dept. of Mathematics and Computer Science,
TU Eindhoven, Eindhoven, The Netherlands
`ocheong@win.tue.nl`

⁴ School of Computing, Soongsil University, Seoul, Korea
`hsnaa@computing.ssu.ac.kr`

⁵ School of Electr. and Inform. Engineering,
Hankuk University of Foreign Studies, Yongin, Korea
`cssin@hufs.ac.kr`

⁶ Dept. of Computer Science, National University of Singapore, Singapore
`antoine@comp.nus.edu.sg`

Abstract. Given a convex polygon P with n vertices, we present algorithms to determine approximations of the largest axially symmetric convex polygon S contained in P , and the smallest such polygon S' that contains P . More precisely, for any $\varepsilon > 0$, we can find an axially symmetric convex polygon $Q \subset P$ with area $|Q| > (1 - \varepsilon)|S|$ in time $O(n + 1/\varepsilon^{3/2})$, and we can find an axially symmetric convex polygon Q' containing P with area $|Q'| < (1 + \varepsilon)|S'|$ in time $O(n + (1/\varepsilon^2) \log(1/\varepsilon))$. If the vertices of P are given in a sorted array, we can obtain the same results in time $O((1/\sqrt{\varepsilon}) \log n + 1/\varepsilon^{3/2})$ and $O((1/\varepsilon) \log n + (1/\varepsilon^2) \log(1/\varepsilon))$ respectively.

1 Introduction

A polygon is *axially symmetric* if it is symmetric with respect to a line. In this paper we present algorithms to determine, for a given convex polygon P , approximations of the largest axially symmetric convex polygon contained in P , and the smallest such polygon that contains P .

^{*} Part of this research was carried out while the authors were participating in the 2nd Korean Workshop on Computational Geometry. The first author acknowledges support from Brain Korea 21 program of MOE. Research of the fourth author is supported by Soongsil University research fund. Research of the fifth author was supported by Hankuk University of Foreign Studies Research Fund of 2004. Research of the last author was supported by the National University of Singapore under grant R-252-000-166-112.

There are a number of papers that study the best inner approximation of any convex set by a symmetric set; the distance to a symmetric set can be considered a measure of its symmetry [7]. Lower bounds for this distance are given by the Löwner-John ellipsoid [8]: any planar convex body C lies between two homothetic ellipses $E \subset C \subset 2E$ with homothety ratio at most 2. Since any ellipse is axially symmetric, and $\text{area}(E) = \frac{1}{4} \text{area}(2E) \geq \frac{1}{4} \text{area}(C)$, any convex planar set C contains an axially symmetric subset with at least $1/4$ of the area of C . The same lower bound of $1/4$ follows from the fact that any planar convex body lies between two homothetic rectangles with homothety ratio at most two [10, 13]. The lower bound can be raised to $2/3$ [9], but that is not known to be tight.

The largest centrally symmetric set contained in a convex shape C is the maximum intersection of C and a translate of $-C$. If C is a convex n -gon, this can be computed in $O(n \log n)$ time [5]. Approximation by axially symmetric sets is technically more demanding, as the largest axially symmetric set contained in C is the maximum intersection of C and a *rotated* and translated copy of C' (with C' a fixed axially reflected copy of C). We do not know of any exact algorithm to compute the maximum intersection of two convex polygons under translation and rotation (orientation-preserving rigid motions), indeed it is not clear that such an algorithm can exist within a reasonable model of computation.

A recent manuscript [4] proposes a practical algorithm to compute exactly the largest subset of a (not necessarily convex) n -gon with an axial symmetry; but it requires to solve $\theta(n^3)$ intractable optimization problems. This is our motivation to give a fast approximation algorithm for the problem. We can find an ε -approximation to that set in time $O(n + 1/\varepsilon^{3/2})$. We also obtain a sublinear time algorithm when the vertices of P are given in a sorted array.

Theorem 1. *Let P be a convex polygon in the plane with n vertices. Given $\varepsilon > 0$, we can find a set $Q \subset P$ with axial symmetry and*

$$\text{area}(Q) > (1 - \varepsilon) \max \left\{ \text{area}(Q^*) \mid Q^* \subseteq P \text{ and } Q^* \text{ axially symmetric} \right\}$$

in time $O(n + 1/\varepsilon^{3/2})$. If the vertices of P are given in a sorted array, we can find Q in time $O((1/\sqrt{\varepsilon}) \log n + 1/\varepsilon^{3/2})$.

The problem of outer approximation of a convex polygon by an axially symmetric polygon seems to have received less interest than inner approximation, perhaps because this is equivalent to the inner approximation problem if one drops the requirement that the axially symmetric polygon has to be convex. The results on approximation by homothetic pairs (ellipses or rectangles) cited above give again simple bounds: for each convex set C there is an axially symmetric set D containing C with $\text{area}(D) \leq 4 \text{area}(C)$. The constant 4 can be reduced to $31/16$ [9], again this is probably not tight. We give a fast approximation algorithm for this problem (the proof will be given in the full version of this paper).

Theorem 2. *Let P be a convex polygon in the plane with n vertices. Given $\varepsilon > 0$, we can find a convex set $Q \supset P$ with axial symmetry and*

$$\text{area}(Q) < (1 + \varepsilon) \min \left\{ \text{area}(Q^*) \mid Q^* \supseteq P \text{ and } Q^* \text{ convex and axially symmetric} \right\}$$

in time $O(n + (1/\varepsilon^2) \log(1/\varepsilon))$. If the vertices of P are given in a sorted array, we can find Q in time $O((1/\varepsilon) \log n + (1/\varepsilon^2) \log(1/\varepsilon))$.

Our algorithms are based on three key ideas.

- First, we approximate the input polygon by a convex polygon whose size depends only on ε . In Theorem 1, we use Dudley’s constructive proof [6]. A recent paper by Agarwal et al. [1] uses this idea as well. This construction does not seem to help in Theorem 2, so we use an approximation in a stronger sense, but with larger size.
- Second, we discretize the set of directions and sample only directions in a discrete set. This works well as long as the polygon is not long and skinny. Fortunately we can show that for long and skinny polygons, the axis of an optimal symmetry must be very close to the diameter of the polygon, or must be nearly orthogonal to this diameter.
- Finally, we use an algorithm to compute the optimal solution for a given direction of the axis of symmetry. In Theorem 1, this is equivalent to finding the translation of C' that maximizes the area of $C \cap C'$. As mentioned before, this can be done in time $O(n \log n)$ [5]. In our case, it suffices to consider a one-dimensional set of translations, which permits a linear time solution [3]. This solution makes use of the area of cross-sections of a three-dimensional polytope and the Brunn-Minkowski Theorem. We do not know of a similarly efficient solution for the case of Theorem 2, so we will give a plane sweep algorithm.

2 Approximating a Convex Polygon

A key component of our proofs is an approximation to a given polygon whose size depends only on ε . We will employ two different methods. For the circumscribed case, we use a simple sampling approach (see Lemma 1). For the inscribed case we use Dudley’s method [6], which gives an approximation with fewer vertices, but in a weaker sense (see Lemma 2). This partly explains why the running time of our algorithm for the inscribed case has a lower dependency on $1/\varepsilon$.

Here and in the rest of this paper, $|P|$ denotes the *area* of a polygon P . For two sets A and B such that $A \subset B$, the *Hausdorff distance* between A and B is

$$\max_{b \in B} \left(\min_{a \in A} \|a - b\| \right)$$

where $\|a - b\|$ is the Euclidean distance between a and b .

Lemma 1. *Given a convex n -gon P and $\varepsilon > 0$, one can construct in time $O(n)$ a convex polygon $P_\varepsilon^h \subset P$ with $O(1/\varepsilon)$ vertices such that the Hausdorff distance between P and P_ε^h is less than $\varepsilon|P|/30 \text{ diam}(P)$. If the vertices of P are given in a sorted array, we can find P_ε^h in time $O((1/\varepsilon) \log n)$.*

Proof. Given the convex n -gon P , we first determine a pair $\{a, b\}$ of diametral vertices of P in $O(n)$ time ([12] p. 176), and then find the width of P orthogonal

to ab , also in $O(n)$ time. This results in a rectangle R , one side of which is parallel to ab and has length $\text{diam}(P)$, and P touches all four sides of R . We use an orthonormal basis where the x -axis is parallel to ab . Intuitively, we will select $O(1/\varepsilon)$ vertices of P that are evenly spaced with respect to their y -coordinate.

The boundary of P consists of two y -monotone chains C_l and C_r . We select vertices of these chains to form the set of vertices of P_ε^h . First we select the lowest vertex (that is, the vertex with smallest y -coordinate). Then we traverse C_l (resp. C_r) upward. After selecting any vertex v , we select the next vertex v' as follows:

- if the vertex v'' that immediately follows v along C_l (resp. C_r) is such that $y(v'') - y(v) > \varepsilon|P|/30 \text{diam}(P)$ then select $v' = v''$.
- otherwise v' is the highest vertex of C_l (resp. C_r) such that $y(v') - y(v) \leq \varepsilon|P|/30 \text{diam}(P)$.

This ensures that any two consecutive vertices of P_ε^h either are consecutive along P , or their vertical distance is at most $\varepsilon|P|/30 \text{diam}(P)$. So the Hausdorff distance between P and P_ε^h is less than $\varepsilon|P|/30 \text{diam}(P)$.

The area of R is at most $2|P|$, so the width of R is at most $2|P|/\text{diam}(P)$. Also note that, after two steps of the above construction, the y -coordinate has increased by at least $\varepsilon|P|/30 \text{diam}(P)$. Therefore P_ε^h has $O(1/\varepsilon)$ vertices. Clearly, this construction allows us to compute P_ε^h in $O(n)$ time. The sublinear time algorithm when the vertices of P are sorted in an array will be given in the full version of this paper. \square

Lemma 2. *Given a convex n -gon P and $\varepsilon > 0$, one can construct in time $O(n)$ two convex polygons P_ε and P'_ε with $O(1/\sqrt{\varepsilon})$ vertices, such that $P_\varepsilon \subset P \subset P'_\varepsilon$, and $|P'_\varepsilon \setminus P_\varepsilon| < \frac{1}{6}\varepsilon|P|$. If the vertices of P are given in a sorted array, we can find P'_ε in time $O((1/\sqrt{\varepsilon}) \log n)$.*

Proof. As in the proof of Lemma 1, we first find in time $O(n)$ a rectangle R , one side of which is parallel to a diametral pair ab , and P touches all four sides of R . Since the claim of the lemma is invariant under affine transformations we can apply a transformation that maps R to the unit square; thus in the following we will assume that P is inscribed in the unit square, and touches all its four sides.

We go once around P , starting at an arbitrary vertex, and select a subset of the edges of the polygon. We always choose the first edge. Let now $e = uv'$ be the most recently chosen edge, let $e' = vv'$ be the next candidate edge, and let $e'' = v'v''$ be the edge following e' . We choose e' if

- the distance $d(u', v') > \sqrt{\varepsilon}$, or
- the outer normals of e and e'' make an angle larger than $\sqrt{\varepsilon}$.

This process results in a sequence of polygon edges, once around the polygon. We observe that the number of edges selected is $O(1/\sqrt{\varepsilon})$. The perimeter of the polygon P is $O(1)$ (it is inscribed in the unit square), so only $O(1/\sqrt{\varepsilon})$ edges can be chosen according to the first rule. The total change of the outer normal angles is $2\pi = O(1)$, so only $O(1/\sqrt{\varepsilon})$ edges can be chosen according to the second rule.

Let P_ε be the convex hull of these selected edges, and P'_ε be the polygon obtained by extending the selected edges until they form a convex polygon. Then $P_\varepsilon \subset P \subset P'_\varepsilon$.

It remains to bound $|P'_\varepsilon \setminus P_\varepsilon|$. The difference consists of $O(1/\sqrt{\varepsilon})$ triangles $uu'v$, where u and u' are vertices of P , and v is the intersection of the lines supporting two consecutive chosen edges. By the first edge selection criterion, the distance $d(u, u') \leq \sqrt{\varepsilon}$. By the second criterion, the lines uv and $u'v$ make an angle of at most $\sqrt{\varepsilon}$, and so the angle $\angle uvu' \geq \pi - \sqrt{\varepsilon}$. Together this implies that the area of the triangle is at most $\varepsilon^{3/2}$. It follows that $|P'_\varepsilon \setminus P_\varepsilon| = O(\varepsilon)$. The area of P , on the other hand, is at least $1/2$, since P is a convex set inscribed in the unit square, touching all sides and with a diameter parallel to one side. This shows that $|P'_\varepsilon \setminus P_\varepsilon| = O(\varepsilon|P|)$, which proves the lemma. The sublinear time algorithm when the vertices of P are sorted in an array will be given in the full version of this paper. \square

3 Discretizing the Rotation

To find the optimal axis of symmetry, we will test a discrete set of orientations. In this section we bound the error incurred by this discretization. We start by analyzing the change in a convex set being rotated. (A proof of this lemma will be given in the full version of this paper.)

Lemma 3. *Let C be a convex set in the plane, and let C' be a copy of C , rotated by an angle δ around a point p in C . Then $|C \cap C'| \geq |C| - \pi\delta \text{diam}(C)^2$.*

The occurrence of $\text{diam}(C)^2$ instead of $|C|$ is a problem. In the following sections, we will need to argue that if the set is long and skinny, that is, when $\text{diam}(P)^2$ is much larger than $|P|$, we need to sample only directions almost parallel to the diameter.

In the following we denote by $\text{refl}(\cdot, \ell)$ the reflection at line ℓ , so that $\text{refl}(C, \ell)$ is the reflected image of C under reflection at ℓ .

Lemma 4. *Let ℓ and ℓ' be two lines intersecting in a point p with an angle δ , and let P be a convex set. If $p \in P \cap \text{refl}(P, \ell)$, then*

$$|P \cap \text{refl}(P, \ell')| \geq |P \cap \text{refl}(P, \ell)| - 2\pi\delta \text{diam}(P)^2.$$

Proof. The concatenation of the reflection at ℓ and the reflection at ℓ' is a rotation around p by the angle 2δ . Let $Q := P \cap \text{refl}(P, \ell)$. Since Q is symmetric with respect to ℓ , the set $\text{refl}(Q, \ell') = \text{refl}(\text{refl}(Q, \ell), \ell')$ is a copy of Q rotated around p by 2δ . Since $p \in Q$, Lemma 3 implies that

$$|Q \cap \text{refl}(Q, \ell')| \geq |Q| - 2\pi\delta \text{diam}(Q)^2.$$

Since $Q \subset P$, we have

$$P \cap \text{refl}(P, \ell') \supset Q \cap \text{refl}(Q, \ell'),$$

and by the above that implies

$$|P \cap \text{refl}(P, \ell')| \geq |Q| - 2\pi\delta \text{diam}(Q)^2 \geq |P \cap \text{refl}(P, \ell)| - 2\pi\delta \text{diam}(P)^2.$$

\square

A similar bound holds for $\text{conv}(P \cup \text{refl}(P, \ell))$.

Lemma 5. *Let ℓ and ℓ' be two lines intersecting in a point p with an angle δ , and let P be a convex set. If $p \in P$, then*

$$|\text{conv}(P \cup \text{refl}(P, \ell'))| \leq |\text{conv}(P \cup \text{refl}(P, \ell))| + 4\pi(1 + \pi/2)\delta \text{diam}(P)^2 .$$

Proof. Let $Q := \text{conv}(P \cup \text{refl}(P, \ell))$ and $Q' := \text{conv}(P \cup \text{refl}(P, \ell'))$. As in Lemma 3, we argue that any point of $\text{refl}(P, \ell')$ has distance at most $2\delta \text{diam}(P)$ from some point of $\text{refl}(P, \ell)$. This implies that Q' is contained in the Minkowski-sum of Q with a disk of radius $2\delta \text{diam}(P)$. This Minkowski-sum has area

$$|Q| + 2\delta \text{diam}(P) \text{peri}(Q) + \pi(2\delta \text{diam}(P))^2 .$$

Since $p \in P$, we have $P \cap \text{refl}(P, \ell) \neq \emptyset$, and so $\text{diam}(Q) \leq 2 \text{diam}(P)$. This implies $\text{peri}(Q) \leq 2\pi \text{diam}(P)$, and we obtain

$$|Q'| \leq |Q| + 4\pi(\delta + \delta^2) \text{diam}(P)^2 \leq |Q| + 4\pi(1 + \pi/2)\delta \text{diam}(P)^2 .$$

□

4 The Largest Axially Symmetric Inscribed Set

Let C be a convex set in the plane and let ℓ be a line. For any line ℓ the set $C \cap \text{refl}(C, \ell)$, if it is not empty, is an axially symmetric convex subset of C , the largest axially symmetric subset with reflection axis ℓ .

Our goal is to find, for a convex polygon P , the line $\ell^{\text{opt}}(P)$ that maximizes the area of this set:

$$\left| P \cap \text{refl}(P, \ell^{\text{opt}}(P)) \right| = \max_{\ell \in \mathbb{R}^2} \left| P \cap \text{refl}(P, \ell) \right| .$$

As we discussed in the introduction, Lassak proved the following lower bound [9]:

Lemma 6.

$$\left| P \cap \text{refl}(P, \ell^{\text{opt}}(P)) \right| \geq \frac{2}{3} |P|$$

Theorem 1 claims that at least an ε -approximation ℓ_ε with

$$(1 - \varepsilon) \left| P \cap \text{refl}(P, \ell^{\text{opt}}(P)) \right| < \left| P \cap \text{refl}(P, \ell_\varepsilon) \right|$$

can be found fast. Our proof of Theorem 1 relies on Lemma 2, Lemma 6 and the following two lemmas.

Lemma 7. *Given a convex n -gon P and a line ℓ , one can find in $O(n)$ time the line ℓ' parallel to ℓ that maximizes $\left| P \cap \text{refl}(P, \ell') \right|$.*

Proof. Let $Q := \text{refl}(P, \ell)$, and let t be a vector orthogonal to ℓ . For any line ℓ' parallel to ℓ , $\text{refl}(P, \ell')$ is the translation of Q by a multiple of t , and so the problem is equivalent to finding the $\lambda \in \mathbb{R}$ such that $|P \cap (Q + \lambda t)|$ is maximized. A linear-time algorithm to solve this problem has been given by Avis et al. [3].

□

We will apply this algorithm to a set of $O(1/\varepsilon)$ direction. The following proof shows how to find this set. Intuitively, when P is fat we will just sample the space of directions uniformly. When P is long and skinny, we will sample more densely, but we will only sample near the two axis of symmetry of a bounding rectangle R that is parallel to a diametral segment ab (see Fig. 1).

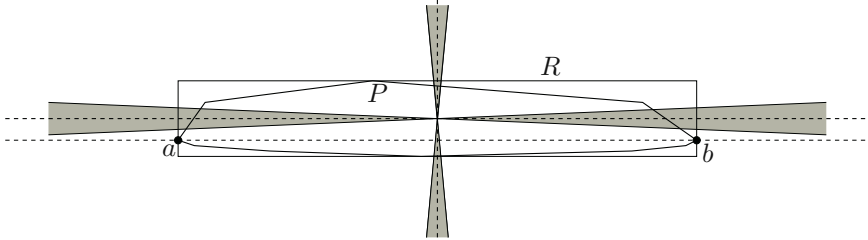


Fig. 1. Case where P is long and skinny. We sample directions from the shaded area.

Lemma 8. *Given a convex n -gon P and $\varepsilon > 0$, one can construct in time $O(n + 1/\varepsilon)$ a set D_ε of $O(1/\varepsilon)$ directions such that*

$$(1 - \tfrac{1}{2}\varepsilon) \left| P \cap \text{refl}(P, \ell^{\text{opt}}(P)) \right| \leq \max \left\{ \left| P \cap \text{refl}(P, \ell) \right| : \ell \text{ has a direction from } D_\varepsilon \right\} .$$

Proof. By Lemma 4 and Lemma 6 it is sufficient to choose the set D_ε such that it contains a line that makes an angle δ of at most $\varepsilon|P|/6\pi \text{diam}(P)^2$ with ℓ^{opt} .

We start by computing, in time $O(n)$, a diameter ab of P , and the area $|P|$. We then distinguish two cases.

If $\text{diam}(P)^2 \leq 20|P|$, then we generate D_ε by sampling the direction space uniformly, choosing multiples of $\varepsilon/60\pi$. Since $\varepsilon/120\pi \leq \varepsilon|P|/6\pi \text{diam}(P)^2$, this is sufficient.

If, on the other hand, $\text{diam}(P)^2 > 20|P|$, then we sample uniformly the directions within $3\pi|P|/2 \text{diam}(P)^2$ of the direction of the diameter ab , choosing multiples of $\varepsilon|P|/3\pi \text{diam}(P)^2$. We do the same around the direction that is orthogonal to ab . To show that this is sufficient we have to demonstrate that $\ell^{\text{opt}}(P)$ does not make an angle larger than $3\pi|P|/2 \text{diam}(P)^2$ with the direction of the diameter or with the direction that is orthogonal to the diameter.

As in the proof of Lemma 2, we can choose a rectangle R parallel to ab of length $\text{diam}(P)$ such that P touches all four sides of R . This implies that the area of R is at most $2|P|$, and so its width is at most $2|P|/\text{diam}(P)$. It follows that P lies in an infinite strip of width at most $2|P|/\text{diam}(P)$. Let $\gamma \in [0, \pi]$ be the angle made by the lines $\ell^{\text{opt}}(P)$ and ab . The set $\text{refl}(P, \ell^{\text{opt}})$ is contained in a congruent strip, intersecting the strip of P at an angle 2γ . The set $P \cap \text{refl}(P, \ell^{\text{opt}})$ is contained in the intersection of the strips, which has area at most $4|P|^2/\text{diam}(P)^2 |\sin 2\gamma|$. By Lemma 6, we know that $|P \cap \text{refl}(P, \ell^{\text{opt}}(P))| \geq 2|P|/3$, so the angle γ must satisfy

$$\frac{4|P|^2}{\text{diam}(P)^2 |\sin 2\gamma|} \geq \frac{2|P|}{3} ,$$

thus $|\sin 2\gamma| \leq 6|P|/\text{diam}(P)^2$. It means that we are in one of the following three cases: $\gamma \leq 3\pi|P|/2\text{diam}(P)^2$, or $\pi - \gamma \leq 3\pi|P|/2\text{diam}(P)^2$, or $|\gamma - \pi/2| \leq 3\pi|P|/2\text{diam}(P)^2$. \square

We can now complete the proof of Theorem 1. Given the convex polygon P and $\varepsilon > 0$, we construct the outer approximating polygon P'_ε of Lemma 2, obtain for this polygon a set of $O(1/\varepsilon)$ directions from Lemma 8, and determine for each of them the optimal line by Lemma 7.

It takes time $O(n)$ to construct P'_ε , or even $O((1/\varepsilon) \log n)$ when the vertices of P are given in a sorted array. It takes time $O(1/\sqrt{\varepsilon} + 1/\varepsilon) = O(1/\varepsilon)$ to construct D_ε , and for each of the $O(1/\varepsilon)$ directions it takes time $O(1/\sqrt{\varepsilon})$ to find the optimal line of that direction. Together this is the claimed complexity of $O(n + 1/\varepsilon^{3/2})$, or $O((1/\sqrt{\varepsilon}) \log n + 1/\varepsilon^{3/2})$ when the vertices of P are given in a sorted array.

It remains to show that the line ℓ_ε with the largest intersection gives the approximation claimed by Theorem 1.

$$\begin{aligned} (1 - \tfrac{1}{2}\varepsilon) \left| P \cap \text{refl}(P, \ell^{\text{opt}}(P)) \right| &< (1 - \tfrac{1}{2}\varepsilon) \left| P'_\varepsilon \cap \text{refl}(P'_\varepsilon, \ell^{\text{opt}}(P)) \right| \\ &< (1 - \tfrac{1}{2}\varepsilon) \left| P'_\varepsilon \cap \text{refl}(P'_\varepsilon, \ell^{\text{opt}}(P'_\varepsilon)) \right| \\ &< \left| P'_\varepsilon \cap \text{refl}(P'_\varepsilon, \ell_\varepsilon) \right| \\ &< \left| P \cap \text{refl}(P, \ell_\varepsilon) \right| + 2|P'_\varepsilon \setminus P| \\ &< \left| P \cap \text{refl}(P, \ell_\varepsilon) \right| + \tfrac{1}{3}\varepsilon|P| \\ &\leq \left| P \cap \text{refl}(P, \ell_\varepsilon) \right| + \tfrac{1}{2}\varepsilon \left| P \cap \text{refl}(P, \ell^{\text{opt}}(P)) \right|. \end{aligned}$$

In the last inequality we used Lemma 6. It follows that

$$(1 - \varepsilon) \left| P \cap \text{refl}(P, \ell^{\text{opt}}(P)) \right| < \left| P \cap \text{refl}(P, \ell_\varepsilon) \right|,$$

which completes the proof of Theorem 1.

5 Concluding Remarks

Lopez and Reisner proposed a different algorithm for approximating a convex n -gon by an inscribed polygon with few vertices [11]. They obtain in time $O(n + (n - k) \log n)$ an approximating polygon with k vertices and relative approximation error $O(1/k^2)$. Our method (see Lemma 2) improves on it as we obtain the same approximation error, with the same number of vertices, in time $O(n)$. When the vertices of P are given in a sorted array, our time bound becomes $O(k \log n)$.

As mentioned before, the problem solved by Theorem 1 is a special case of the problem of maximizing the overlap of two convex polygons P and Q under translation and rotation of Q . Surprisingly little is known about this problem. Alt et al. [2] made some initial progress on a similar problem, showing, for instance, how to construct, for a convex polygon P , the axis-parallel rectangle Q minimizing the symmetric difference of P and Q .

Our solution does not generalize to this problem. We make use of two special properties of our problem. First, as discussed before, we know that the area of the optimal solution is within a constant factor the area of P . Second, we have a specific argument if P is very long and skinny.

References

1. P. K. Agarwal, S. Har-Peled, K. R. Varadarajan: Approximating Extent Measures of Points, manuscript 2003.
2. H. Alt, J. Blömer, M. Godau, and H. Wägener: Approximation of convex polygons, *Proc. 17th Internat. Colloq. Automata Lang. Program.*, Lecture Notes Comput. Sci. **443**, p. 703–716, Springer-Verlag 1990.
3. D. Avis, P. Bose, G. Toussaint, T. Shermer, B. Zhu, J. Snoeyink: On the sectional area of convex polytopes, *Proc. 12th ACM Symp. Comput. geometry*, (1996) 411–412.
4. G. Barequet, V. Rogol: Maximizing the area of an axially-symmetric polygon inscribed by a simple polygon, manuscript 2003.
5. M. de Berg, O. Devillers, M. van Kreveld, O. Schwarzkopf, M. Teillaud: Computing the maximum overlap of two convex polygons under translations, *Theory of Computing Systems* **31** (1998) 613–628.
6. R. M. Dudley: Metric entropy of some classes of sets with differentiable boundaries, *J. Approximation Theory* **10** (1974) 227–236; Erratum in *J. Approx. Theory* **26** (1979) 192–193.
7. B. Grünbaum: Measures of symmetry of convex sets, in ‘Convexity’, V. Klee, ed., *Proc. Symp. Pure Math* **7**, Amer. Math. Soc. 1963, 233–270.
8. F. John: Extremum problems with inequalities as subsidiary conditions, in ‘Courant Anniversary Volume’, 1948, 187–204.
9. M. Lassak: Approximation of convex bodies by axially symmetric bodies, *Proc. Amer. Math. Soc.* **130** (2002) 3075–3084. Erratum in *Proc. Amer. Math. Soc.* **131** (2003) p.2301
10. M. Lassak: Approximation of convex bodies by rectangles, *Geometriae Dedicata* **47** (1993) 111–117.
11. M.A. Lopez, S. Reisner: Efficient approximation of convex polygons, *Internat. J. Comput. Geom. Appl.* **10** (2000) 445–452.
12. P. Preparata, M.I. Shamos: Computational Geometry: An Introduction, Springer-Verlag 1985.
13. O. Schwarzkopf, U. Fuchs, G. Rote, E. Welzl: Approximation of convex figures by pairs of rectangles, *Comput. Geom. Theory Appl.* **10** (1998) 77–87; also in STACS 1990, LNCS 415, p. 240–249.
14. B. A. deValcourt: Axially symmetric polygons inscribed in and circumscribed about convex polygons, *Elemente Math.* **22** (1967) 121–133.

The Traveling Salesman Problem with Few Inner Points

Vladimir G. Deĭneko^{1,*}, Michael Hoffmann²,
Yoshio Okamoto^{2, **}, and Gerhard J. Woeginger^{3,*}

¹ Warwick Business School, The University of Warwick,
Conventry CV4 7AL, United Kingdom
orsvd@wbs.warwick.ac.uk

² Institute of Theoretical Computer Science, ETH Zurich, CH-8092 Zurich, Switzerland
{hoffmann, okamoto}@inf.ethz.ch

³ Department of Mathematics and Computer Science, TU Eindhoven,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
gwoegi@win.tue.nl

Abstract. We study the traveling salesman problem (TSP) in the 2-dimensional Euclidean plane. The problem is NP-hard in general, but trivial if the points are in convex position. In this paper, we investigate the influence of the number of inner points (i.e., points in the interior of the convex hull) on the computational complexity of the problem. We give two simple algorithms for this problem. The first one runs in $O(k!kn)$ time and $O(k)$ space, and the second runs in $O(2^k k^2 n)$ time and $O(2^k kn)$ space, when n is the total number of input points and k is the number of inner points. Hence, if k is taken as a parameter, this problem is fixed-parameter tractable (FPT), and also can be solved in polynomial time if $k = O(\log n)$. We also consider variants of the TSP such as the prize-collecting TSP and the partial TSP in this setting, and show that they are FPT as well.

1 Introduction

The traveling salesman problem (TSP) is one of the most famous optimization problems, which comes along many kinds of applications such as logistics, scheduling, VLSI manufacturing, etc. In many practical applications, we have to solve TSP instances arising from the two-dimensional Euclidean plane, which we call the 2DTSP. Also most of the benchmark instances for TSP belong to this class. Theoretically speaking, the general 2DTSP is strongly NP-hard [9, 15]. On the other hand, the problem is trivial if the points are in convex position, i.e., they are the vertices of a convex polygon. Therefore, the following natural question is asked: What is the influence of the number of inner points on the complexity of the problem? Here, an inner point of a finite point set P is a point from P which lies in the interior of the convex hull of P . Intuitively, we might say that “Fewer inner points make the problem easier to solve.”

In the following, we answer this question and support the intuition above by providing simple algorithms based on the dynamic programming paradigm. The first one runs

* Supported by the NWO project 613.000.322 “Exact Algorithms”.

** Supported by the Berlin-Zurich Joint Graduate Program “Combinatorics, Geometry, and Computation” (CGC), financed by ETH Zurich and the German Science Foundation (DFG).

in $O(k!kn)$ time and $O(k)$ space, and the second runs in $O(2^k k^2 n)$ time and $O(2^k kn)$ space, where n is the total number of input points and k is the number of inner points. Hence, from the viewpoint of parameterized complexity [7, 14], these algorithms are fixed-parameter algorithms¹, when the number of inner points is taken as a parameter, hence the problem is fixed-parameter tractable (FPT). Observe that the second algorithm gives a polynomial-time solution to the problem when $k = O(\log n)$.

We also study two variants of the traveling salesman problem, both also strongly NP-hard: the prize-collecting traveling salesman problem, introduced by Balas [2], and the partial traveling salesman problem. We show that these problems on the Euclidean plane are FPT as well.

Related work. Since the literature on the TSP and its variants is vast, we only point out studies on the TSP itself which are closely related to our result. To the authors' knowledge, only few papers studied the parameterized complexity of the 2DTSP. Probably the most closely related one is a paper by Deĭneko, van Dal and Rote [5]. They studied the 2DTSP where the inner points lie on a line. The problem is called the *convex-hull-and-line TSP*. They gave an algorithm running in $O(kn)$ time, where k is the number of inner points. Deĭneko and Woeginger [6] studied a slightly more general problem called the *convex-hull-and- ℓ -line TSP*, and gave an algorithm running in $O(k^\ell n^2)$ time. Compared to these results, our algorithms deal with the most general situation, and are fixed-parameter algorithms with respect to k . As for approximation algorithms, Arora [1] and Mitchell [13] found polynomial-time approximation schemes (PTAS) for the 2DTSP. Rao and Smith [16] gave a PTAS with better running time $O(n \log n + 2^{\text{poly}(1/\varepsilon)} n)$. As for exact algorithms, Held and Karp [10] and independently Bellman [3] provided a dynamic programming algorithm to solve the TSP optimally in $O(2^n n^2)$ time and $O(2^n n)$ space. For geometric problems, Hwang, Chang and Lee [12] gave an algorithm to solve the 2DTSP in $O(n^{O(\sqrt{n})})$ time based on the so-called separator theorem.

Organization. The next section introduces the problem formally, and gives a fundamental lemma. Sect. 3 and 4 describe algorithms for the 2DTSP. Variations are discussed in Sect. 5. We conclude with an open problem at the final section.

2 Traveling Salesman Problem with Few Inner Points

Let $P \subseteq \mathbb{R}^2$ be a finite point set in the Euclidean plane. The *convex hull* of P is the smallest convex set containing P . A point $p \in P$ is called an *inner* point if p lies in the interior of the convex hull of P . We denote by $\text{Inn}(P)$ the set of inner points of P . A point $p \in P$ is called an *outer* point if it is not an inner point, i.e., it is on the boundary of the convex hull of P . We denote by $\text{Out}(P)$ the set of outer points of P . Note that $P = \text{Inn}(P) \cup \text{Out}(P)$ and $\text{Inn}(P) \cap \text{Out}(P) = \emptyset$. Let $n := |P|$ and $k := |\text{Inn}(P)|$. (So, we have $|\text{Out}(P)| = n - k$.)

¹ A *fixed-parameter algorithm* has running time $O(f(k)\text{poly}(n))$, where n is the input size, k is a parameter and $f : \mathbb{N} \rightarrow \mathbb{N}$ is an arbitrary function. For example, an algorithm with running time $O(440^k n)$ is a fixed-parameter algorithm whereas one with $O(n^k)$ is not.

A *tour* on P is a linear order (x_1, x_2, \dots, x_n) of the points in P . We say that this tour *starts at* x_1 . We often identify the tour (x_1, \dots, x_n) on P with a closed polygonal curve consisting of the line segments $\overline{x_1x_2}, \overline{x_2x_3}, \dots, \overline{x_{n-1}x_n}, \overline{x_nx_1}$. The *length* of the tour is the Euclidean length of this polygonal curve, i.e., $\sum_{i=1}^{n-1} d(x_i, x_{i+1}) + d(x_n, x_1)$, where $d(x_i, x_{i+1})$ stands for the Euclidean distance from x_i to x_{i+1} . The objective of the TSP is to find a shortest tour. The following lemma was probably first noted by Flood [8] and nowadays it is folklore.

Lemma 1 (Flood [8]). *Every shortest tour has no crossing.*

This lemma immediately implies the following lemma, which plays a fundamental role in our algorithm. We call a linear order on $\text{Out}(P)$ *cyclic* if every two consecutive points in the order are also consecutive on the convex hull of P .

Lemma 2. *In every shortest tour on P , the points of $\text{Out}(P)$ appear in a cyclic order.*

With Lemma 2, we can establish the following naive algorithm: take an arbitrary cyclic order on $\text{Out}(P)$, then look through all tours (i.e., the linear orders) π on P which respect² this cyclic order. Compute the length of each tour, and output the best one among them. The number of such tours is $O(k!n^k)$. Therefore, the running time of this algorithm is $O(k!n^{k+1})$. So, if k is constant, this algorithm runs in polynomial time. However, it is not a fixed-parameter algorithm with respect to k since k appears in the exponent of n .

3 First Fixed-Parameter Algorithm

First, let us remark that later on we always assume that, when P is given to an algorithm as input, $\text{Out}(P)$ is distinguished together with a cyclic order $\gamma = (p_1, \dots, p_{n-k})$. Also, note that the space complexity in the algorithms below do not count the input size, as usual in theoretical computer science.

Our first algorithm uses the following idea. We look through all linear orders on $\text{Inn}(P)$. Let us first fix a linear order π on $\text{Inn}(P)$. We will try to find a shortest tour on P which respects both the cyclic order γ on $\text{Out}(P)$ and the linear order π on $\text{Inn}(P)$. Then, we exhaust this procedure for all linear orders on $\text{Inn}(P)$, and output a minimum one. Later we will show that we can compute such a tour in $O(kn)$ time and $O(k)$ space. Then, since the number of linear orders on $\text{Inn}(P)$ is $k!$ and they can be enumerated in amortized $O(1)$ time per one with $O(k)$ space [17], overall the algorithm runs in $O(k!kn)$ time and $O(k)$ space.

Now, given a cyclic order γ on $\text{Out}(P)$ and a linear order π on $\text{Inn}(P)$, we describe how to compute a shortest tour among those which respect γ and π by dynamic programming. For dynamic programming in algorithmics, see the textbook by Cormen, Leiserson, Rivest and Stein [4], for example.

We consider a three-dimensional array $F_1[i, j, m]$, where $i \in \{1, \dots, n-k\}$, $j \in \{0, 1, \dots, k\}$, and $m \in \{\text{Inn}, \text{Out}\}$. The first index i represents the point p_i in $\text{Out}(P)$,

² For a set S and a subset $S' \subseteq S$, we say a linear order π on S *respects* a linear order π' on S' if the restriction of π onto S' is π' .

the second index j represents the point q_j in $\text{Inn}(P)$, and the third index m represents the position. The value $F_1[i, j, m]$ represents the length of a shortest “path” on $\{p_1, \dots, p_i\} \cup \{q_1, \dots, q_j\}$ that satisfies the following conditions.

- It starts at $p_1 \in \text{Out}(P)$.
- It visits exactly the points in $\{p_1, \dots, p_i\} \cup \{q_1, \dots, q_j\}$. (If $j = 0$, set $\{q_1, \dots, q_j\} = \emptyset$.)
- It respects the orders γ and π .
- If $m = \text{Out}$, then it ends at p_i (an outer point). If $m = \text{Inn}$, then it ends at q_j (an inner point).

Then, the length of a shortest tour respecting π and γ can be computed as

$$\min\{F_1[n-k, k, \text{Out}] + d[p_{n-k}, p_1], F_1[n-k, k, \text{Inn}] + d(q_k, p_1)\}.$$

Therefore, it suffices to know the values $F_1[i, j, m]$ for all possible i, j, m .

To do that, we establish a recurrence. First let us look at the boundary cases.

- Since we start at p_1 , set $F_1[1, 0, \text{Out}] = 0$.
- There are some impossible states for which we set the values to ∞ . Namely, for every $j \in \{1, \dots, k\}$ set $F_1[1, j, \text{Out}] = \infty$, and for every $i \in \{1, \dots, n-k\}$ set $F_1[i, 0, \text{Inn}] = \infty$.

Now, assume we want to visit the points of $\{p_1, \dots, p_i\} \cup \{q_1, \dots, q_j\}$ while respecting the orders γ and π and arrive at q_j . How can we get to this state? Since we respect the orders, either (1) first we have to visit the points of $\{p_1, \dots, p_i\} \cup \{q_1, \dots, q_{j-1}\}$ to arrive at p_i then move to q_j , or (2) first we have to visit the points of $\{p_1, \dots, p_i\} \cup \{q_1, \dots, q_{j-1}\}$ to arrive at q_{j-1} then move to q_j . Therefore, we have

$$F_1[i, j, \text{Inn}] = \min\{F_1[i, j-1, \text{Out}] + d(p_i, q_j), F_1[i, j-1, \text{Inn}] + d(q_{j-1}, q_j)\} \quad (1)$$

for $(i, j) \in \{1, \dots, n-k\} \times \{1, \dots, k\}$. Similarly, we have

$$F_1[i, j, \text{Out}] = \min\{F_1[i-1, j, \text{Out}] + d(p_{i-1}, p_i), F_1[i-1, j, \text{Inn}] + d(q_j, p_i)\} \quad (2)$$

for $(i, j) \in \{2, \dots, n-k\} \times \{0, \dots, k\}$, where $d(q_0, p_i)$ is considered ∞ for convenience. Since what is referred to in the right-hand sides of Equation (1) and (2) has smaller indices, we can solve this recursion in a bottom-up way by dynamic programming. This completes the dynamic-programming formulation for the computation of $F_1[i, j, m]$.

The size of the array is $(n-k) \times (k+1) \times 2 = O(kn)$, and the computation of each entry requires to look up at most two other entries of the array. Therefore, we can fill up the array in $O(kn)$ time and $O(kn)$ space.

Now, we describe how to reduce the space requirement to $O(k)$. For each $(i, j) \in \{1, \dots, n-k\} \times \{1, \dots, k\}$, consider when $F_1[i, j, \text{Inn}]$ is looked up throughout the computation. It is looked up only when we compute $F_1[i, j+1, \text{Inn}]$ and $F_1[i+1, j, \text{Out}]$. So the effect of $F_1[i, j, \text{Inn}]$ is local. Similarly, the value $F_1[i, j, \text{Out}]$ is looked up only when we compute $F_1[i, j+1, \text{Inn}]$ and $F_1[i+1, j, \text{Out}]$. We utilize this locality in the computation.

We divide the computation process into some phases. For every $i \in \{1, \dots, n-k\}$, in the i -th phase, we compute $F_1[i, j, \text{Inn}]$ and $F_1[i, j, \text{Out}]$ for all $j \in \{0, \dots, k\}$. Within the i -th phase, the computation start with $F_1[i, 1, \text{Out}]$ and proceed along $F_1[i, 2, \text{Out}], F_1[i, 3, \text{Out}], \dots$, until we get $F_1[i, k, \text{Out}]$. Then, we start calculating $F_1[i, 1, \text{Inn}]$ and proceed along $F_1[i, 2, \text{Inn}], F_1[i, 3, \text{Inn}], \dots$, until we get $F_1[i, k, \text{Inn}]$. From the observation above, all the computation in the i -th phase only needs the outcome from the $(i-1)$ -st phase and the i -th phase itself. Therefore, we only have to store the results from the $(i-1)$ -st phase for each i . This requires only $O(k)$ storage.

In this way, we obtain the following theorem.

Theorem 1. *The 2DTSP on n points including k inner points can be solved in $O(k!kn)$ time and $O(k)$ space. In particular, it can be solved in polynomial time if $k = O(\log n / \log \log n)$. \square*

4 Second Fixed-Parameter Algorithm with Better Running Time

To obtain a faster algorithm, we make use of the trade-off between the time complexity and the space complexity. Compared to the first algorithm, the second algorithm has a better running time $O(2^k k^2 n)$ and needs a larger space $O(2^k kn)$. The idea of trade-off is also taken by the dynamic programming algorithm for the general traveling salesman problem due to Bellman [3] and Held & Karp [10], and our second algorithm is essentially a generalization of their algorithms.

In the second algorithm, first we fix a cyclic order γ on $\text{Out}(P)$. Then, we immediately start the dynamic programming. This time, we consider the following three-dimensional array $F_2[i, S, r]$, where $i \in \{1, \dots, n-k\}$, $S \subseteq \text{Inn}(P)$, and $r \in S \cup \{p_i\}$. We interpret $F_2[i, S, r]$ as the length of a shortest path on $\{p_1, \dots, p_i\} \cup S$ that satisfies the following conditions.

- It starts at $p_1 \in \text{Out}(P)$.
- It visits exactly the points in $\{p_1, \dots, p_i\} \cup S$.
- It respects the order γ .
- It ends at r .

Then, the length of a shortest tour can be computed as

$$\min\{F_2[n-k, \text{Inn}(P), r] + d(r, p_1) \mid r \in \text{Inn}(P) \cup \{p_{n-k}\}\}.$$

Therefore, it suffices to know the values $F_2[i, S, r]$ for all possible triples (i, S, r) .

To do that, we establish a recurrence. The boundary cases are as follows.

- Since we start at p_1 , set $F_2[1, \emptyset, p_1] = 0$.
- Set $F_2[1, S, r] = \infty$ when $S \neq \emptyset$, since this is an unreachable situation.

Let $i \in \{2, \dots, n-k\}$ and $S \subseteq \text{Inn}(P)$. We want to visit the points of $\{p_1, \dots, p_i\} \cup S$ while respecting the order γ and arrive at p_i . How can we get to this state? Since we respect the order γ , we first have to visit the points of $\{p_1, \dots, p_{i-1}\} \cup S$ to arrive at a point in $S \cup \{p_{i-1}\}$ and then move to p_i . Therefore, we have

$$F_2[i, S, p_i] = \min_{t \in S \cup \{p_{i-1}\}} (F_2[i-1, S, t] + d(t, p_i)) \quad (3)$$

for $i \in \{2, \dots, n-k\}$ and $S \subseteq \text{Inn}(P)$. Similarly, we have

$$F_2[i, S, r] = \min_{t \in (S \setminus \{r\}) \cup \{p_i\}} (F_2[i, S \setminus \{r\}, t] + d(t, r)) \quad (4)$$

for $i \in \{2, \dots, n-k\}$, $S \subseteq \text{Inn}(P)$, $S \neq \emptyset$ and $r \in S$. This completes the dynamic-programming formulation for the computation of $F_2[i, S, r]$.

The size of the array in this algorithm is $(n-k) \sum_{s=0}^k \binom{k}{s} s = O(2^k k n)$, and the computation of each entry requires to look up $O(k)$ other entries. Therefore, we can fill up the array in $O(2^k k^2 n)$ time and in $O(2^k k n)$ space. Thus, we obtain the following theorem.

Theorem 2. *The 2DTSP on n points including k inner points can be solved in $O(2^k k^2 n)$ time and $O(2^k k n)$ space. In particular, it can be solved in polynomial time if $k = O(\log n)$. \square*

5 Variants of the Traveling Salesman Problem

Since our approach to the TSP in the previous section is based on the general dynamic programming paradigm, it is also applicable to other variants of the TSP. In this section, we discuss two of them.

5.1 Prize-Collecting Traveling Salesman Problem

In the *prize-collecting TSP*, we are given an n -point set $P \subseteq \mathbb{R}^2$ with a distinguished point $h \in P$ called the *home*, and a non-negative number $c(p) \in \mathbb{R}$ for each point $p \in P$ which we call the *penalty* of p . The goal is to find a subset $P' \subseteq P \setminus \{h\}$ and a tour on $P' \cup \{h\}$ starting at h which minimizes the length of the tour minus the penalties over all $p \in P' \cup \{h\}$. In this section, the value of a tour (or a path) refers to the value of this objective.

For this problem, we basically follow the same procedure as the TSP, but a little attention has to be paid because in this case we have to *select* some of the points from P . In addition, we have to care about the position of h , in $\text{Inn}(P)$ or in $\text{Out}(P)$.

First Algorithm. First, let us consider the case $h \in \text{Out}(P)$. In this case, we fix a cyclic order γ on $\text{Out}(P)$, which starts with h , and we look through all linear orders on $\text{Inn}(P)$. Let $\gamma = (p_1, p_2, \dots, p_{n-k})$, where $p_1 = h$, and fix one linear order $\pi = (q_1, q_2, \dots, q_k)$ on $\text{Inn}(P)$. Then, we consider a three-dimensional array $F_1[i, j, m]$, where $i \in \{1, \dots, n-k\}$, $j \in \{0, 1, \dots, k\}$ and $m \in \{\text{Inn}, \text{Out}\}$. The value $F_1[i, j, m]$ is interpreted as the value of an optimal path on $\{p_1, \dots, p_i\} \cup \{q_1, \dots, q_j\}$ which satisfies the following conditions.

- It starts at $p_1 \in \text{Out}(P)$.
- It visits *some* points from $\{p_1, \dots, p_i\} \cup \{q_1, \dots, q_j\}$, and not more. (If $j = 0$, set $\{q_1, \dots, q_j\} = \emptyset$.)
- It respects the orders γ and π .
- If $m = \text{Out}$, then it ends at p_i . If $m = \text{Inn}$, then it ends at q_j .

We want to compute the values $F_1[i, j, m]$ for all possible triples (i, j, m) .

The boundary cases are: $F_1[1, j, \text{Out}] = -c(p_1)$ for every $j \in \{1, \dots, k\}$; and $F_1[i, 0, \text{Inn}] = \infty$ for every $i \in \{1, \dots, n-k\}$, and the main part of the recurrence is:

$$F_1[i, j, \text{Inn}] = \min\left\{\min_{i' \in \{1, \dots, i\}} (F_1[i', j-1, \text{Out}] + d(p_{i'}, q_j) - c(q_j)), \right. \\ \left. \min_{j' \in \{0, \dots, j-1\}} (F_1[i, j', \text{Inn}] + d(q_{j'}, q_j) - c(q_j))\right\}$$

for $(i, j) \in \{1, \dots, n-k\} \times \{1, \dots, k\}$, and

$$F_1[i, j, \text{Out}] = \min\left\{\min_{i' \in \{1, \dots, i-1\}} (F_1[i', j, \text{Out}] + d(p_{i'}, p_i) - c(p_i)), \right. \\ \left. \min_{j' \in \{0, \dots, j\}} (F_1[i-1, j', \text{Inn}] + d(q_{j'}, p_i) - c(p_i))\right\}$$

for $(i, j) \in \{2, \dots, n-k\} \times \{0, \dots, k\}$. For convenience, $d(q_0, p_i)$ is considered to be ∞ .

Then, the value of an optimal prize-collecting tour respecting π and γ can be computed as

$$\min\left\{\min_{i \in \{1, \dots, n-k\}} (F_1[i, k, \text{Out}] + d(p_i, p_1)), \min_{j \in \{0, \dots, k\}} (F_1[n-k, j, \text{Inn}] + d(q_j, p_1))\right\}.$$

Since the size of the array is $O(kn)$ and each entry can be filled by looking up $O(n)$ other entries, the running time is $O(kn^2)$. Therefore, looking through all linear orders on $\text{Inn}(P)$, the overall running time of the algorithm is $O(k!kn^2)$.

Next, let us consider the case $h \in \text{Inn}(P)$. In this case, we look through all linear orders on $\text{Inn}(P)$ which start with h , and also all cyclic orders on $\text{Out}(P)$. Fix one linear order $\pi = (q_1, q_2, \dots, q_k)$ on $\text{Inn}(P)$, where $q_1 = h$, and one cyclic order $\gamma = (p_1, p_2, \dots, p_{n-k})$ on $\text{Out}(P)$. Then, we consider a three-dimensional array $F_1[i, j, m]$, where $i \in \{0, 1, \dots, n-k\}$, $j \in \{1, \dots, k\}$ and $m \in \{\text{Inn}, \text{Out}\}$. The interpretation and the obtained recurrence is similar to the first case, hence omitted. However, in this case, the number of orders we look through is $O(k!n)$. Therefore, the overall running time of the algorithm in this case is $O(k!kn^3)$. Thus, we obtain the following theorem.

Theorem 3. *The prize-collecting TSP in the Euclidean plane can be solved in $O(k!kn^3)$ time and $O(kn)$ space, when n is the total number of input points and k is the number of inner points. In particular, it can be solved in polynomial time if $k = O(\log n / \log \log n)$. \square*

Second Algorithm. Now we adapt the second algorithm for the 2DTSP to the prize-collecting TSP. Let us consider the case $h \in \text{Out}(P)$. (The case $h \in \text{Inn}(P)$ can be handled in the same way.) For a cyclic order $\gamma = (p_1, \dots, p_{n-k})$ on $\text{Out}(P)$ with $p_1 = h$, we define a three-dimensional array $F_2[i, S, r]$ where $i \in \{1, \dots, n-k\}$, $S \subseteq \text{Inn}(P)$ and $r \in S \cup \{p_i\}$. We interpret $F_2[i, S, r]$ as the value of an optimal path on $\{p_1, \dots, p_i\} \cup S$ that satisfies the following conditions.

- It starts at $p_1 \in \text{Out}(P)$.
- It visits some points of $\{p_1, \dots, p_i\} \cup S$.
- It respects the order γ .
- It ends at r .

Then, the value of an optimal tour can be computed as

$$\min\{F_2[n-k, \text{Inn}(P), r] + d(r, p_1) \mid r \in P\}.$$

The boundary cases are: $F_2[1, \emptyset, p_1] = -c(p_1)$; $F_2[1, S, r] = \infty$ when $S \neq \emptyset$. The main part of the recurrence is

$$F_2[i, S, p_i] = \min_{t \in S \cup \{p_{i-1}\}} (F_2[i-1, S, t] + d(t, p_i) - c(p_i))$$

for $i \in \{2, \dots, n-k\}$ and $S \subseteq \text{Inn}(P)$; and

$$F_2[i, S, r] = \min_{t \in (S \setminus \{r\}) \cup \{p_i\}} (F_2[i, S \setminus \{r\}, t] + d(t, r) - c(r))$$

for $i \in \{2, \dots, n-k\}$, $S \subseteq \text{Inn}(P)$ and $r \in S$. Then, we see that the computation can be done in $O(2^k k^2 n)$ time and $O(2^k k n)$ space.

Theorem 4. *The prize-collecting TSP in the Euclidean plane can be solved in $O(2^k k^2 n)$ time and $O(2^k k n)$ space, when n is the total number of input points and k is the number of inner points. In particular, it can be solved in polynomial time if $k = O(\log n)$. \square*

5.2 Partial Traveling Salesman Problem

In the ℓ -partial TSP³, we are given an n -point set $P \subseteq \mathbb{R}^2$ with a distinguished point $h \in P$ called the home, and a positive integer $\ell \leq n$. We are asked to find a shortest tour starting from h and consisting of ℓ points from P .

Because of space limitations, we do not give an adaptation of the first algorithm for the TSP, although it is possible but more tedious. So, we just describe a variation of the second algorithm.

Second Algorithm. Similarly to the prize-collecting TSP, we distinguish the cases according to the position of h , in $\text{Inn}(P)$ or in $\text{Out}(P)$. Here we only consider the case $h \in \text{Out}(P)$. (The case $h \in \text{Inn}(P)$ is similar.) Fix a cyclic order $\gamma = (p_1, \dots, p_{n-k})$ on $\text{Out}(P)$, where $p_1 = h$. We consider a four-dimensional array $F_2[i, S, r, m]$, where $i \in \{1, \dots, n-k\}$, $S \subseteq \text{Inn}(P)$, $r \in S \cup \{p_i\}$, and $m \in \{1, \dots, \ell\}$. Then, $F_2[i, S, r, m]$ is interpreted as the length of a shortest path that satisfies the following conditions.

- It starts at $p_1 \in \text{Out}(P)$.
- It visits exactly m points of $\{p_1, \dots, p_i\} \cup S$.
- It respects the order γ .
- It ends at r .

Note that the fourth index m represents the number of points which have already been visited. Then, the length of a shortest tour through ℓ points is

$$\min\{F_2[i, \text{Inn}(P), r, \ell] + d(r, p_1) \mid i \in \{1, \dots, n-k\}, r \in \text{Inn}(P) \cup \{p_i\}\}.$$

Therefore, it suffices to compute the values $F_2[i, S, r, m]$ for all possible i, S, r, m .

³ Usually the problem is called the k -partial TSP. However, since k is reserved for the number of inner points in the current work, we will use ℓ instead of k .

The boundary cases are: $F_2[i, S, r, 1] = 0$ if $i = 1$ and $r = p_1$; otherwise $F_2[i, S, r, 1] = \infty$. Also, $F_2[1, S, p_1, m] = \infty$ for $m > 1$. The main part of the recurrence is:

$$F_2[i, S, p_i, m] = \min_{t \in S \cup \{p_{i-1}\}} (F_2[i-1, S, t, m-1] + d(t, p_i))$$

for $i \in \{2, \dots, n-k\}$, $S \subseteq \text{Inn}(P)$ and $m \in \{2, \dots, \ell\}$;

$$F_2[i, S, r, m] = \min_{t \in (S \setminus \{r\}) \cup \{p_i\}} (F_2[i, S \setminus \{r\}, t, m-1] + d(t, r))$$

for $i \in \{1, \dots, n-k\}$, $S \subseteq \text{Inn}(P)$, $r \in S$ and $m \in \{2, \dots, \ell\}$.

Although the size of the array is $O(2^k k \ell n)$, we can reduce the space requirement to $O(2^k k n)$ because of the locality with respect to the fourth index m . In this way, we obtain the following theorem.

Theorem 5. *The ℓ -partial TSP for the Euclidean plane can be solved in $O(2^k k^2 \ell n)$ time and $O(2^k k n)$ space, where n is the total number of input points and k is the number of inner points. In particular, it can be solved in polynomial time if $k = O(\log n)$.*

6 Concluding Remarks

We have investigated the influence of the number of inner points in a two-dimensional geometric problem. Our results support the intuition “Fewer inner points make the problem easier to solve,” and nicely “interpolate” triviality when we have no inner point and intractability for the general case. This interpolation has been explored from the viewpoint of parameterized complexity. Let us remark that the results in this paper can also be applied to the two-dimensional Manhattan traveling salesman problem, where the distance is measured by the ℓ_1 -norm. That is because Lemmata 1 and 2 are also true for that case. More generally, our algorithms solve any TSP instance (not necessarily geometric) for which $n-k$ points have to be visited in a specified order.

To the authors’ knowledge, this work is the first paper that deals with parameterized problems in terms of the number of inner points. Study of other geometric problems within this parameterized framework is an interesting direction of research. Within this framework, the minimum weight triangulation problem was recently studied, and a fixed-parameter algorithm with respect to the number of inner points was given [11].

The major open question is to improve the time complexity $O(2^k k^2 n)$. For example, is there a polynomial-time algorithm for the 2DTSP when $k = O(\log^2 n)$?

Acknowledgments

We are grateful to Emo Welzl for helpful discussions.

References

1. S. Arora: Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *J. ACM* **45** (1998) 753–782.
2. E. Balas: The prize collecting traveling salesman problem. *Networks* **19** (1989) 621–636.

3. R. Bellman: Dynamic programming treatment of the traveling salesman problem. *J. ACM* **9** (1962) 61–63.
4. T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein: Introduction to algorithms, 2nd edition. MIT Press, Cambridge, 2001.
5. V. Deĭneko, R. van Dal and G. Rote: The convex-hull-and-line traveling salesman problem: a solvable case, *Inf. Process. Lett.* **59** (1996) 295–301.
6. V. Deĭneko and G.J. Woeginger. The convex-hull-and- k -line traveling salesman problem. *Inf. Process. Lett.* **59** (1996) 295–301.
7. R.G. Downey and M.R. Fellows: Parameterized Complexity. Springer, Berlin, 1999.
8. M.M. Flood: Traveling salesman problem. *Oper. Res.* **4** (1956) 61–75.
9. M.R. Garey, R.L. Graham and D.S. Johnson: Some NP-complete geometric problems. *Proc. 8th STOC*, 1976, pp. 10–22.
10. M. Held and R. Karp: A dynamic programming approach to sequencing problems. *J. SIAM* **10** (1962) 196–210.
11. M. Hoffmann and Y. Okamoto: The minimum weight triangulation problem with few inner points. Submitted.
12. R.Z. Hwang, R.C. Chang and R.C.T. Lee: The searching over separators strategy to solve some NP-hard problems in subexponential time. *Algorithmica* **9** (1993) 398–423.
13. J. Mitchell: Guillotine subdivisions approximate polygonal subdivisions: a simple polynomial-time approximation scheme for geometric k -MST, TSP, and related problems. *SIAM J. Comput.* **28** (1999) 1298–1309.
14. R. Niedermeier: Invitation to fixed-parameter algorithms. Habilitation Thesis, Universität Tübingen, 2002.
15. C.H. Papadimitriou: Euclidean TSP is NP-complete. *Theor. Comput. Sci.* **4** (1977) 237–244.
16. S. Rao and W. Smith: Approximating geometric graphs via “spanners” and “banyans.” *Proc. 30th STOC*, 1998, pp. 540–550.
17. R. Sedgewick: Permutation generation methods. *ACM Comput. Surveys* **9** (1977) 137–164.

A Faster Algorithm for the All-Pairs Shortest Path Problem and Its Application

Tadao Takaoka

Department of Computer Science
University of Canterbury, Christchurch, New Zealand
`tad@cosc.canterbury.ac.nz`

Abstract. We design a faster algorithm for the all-pairs shortest path problem under the RAM model, based on distance matrix multiplication (DMM). Specifically we improve the best known time complexity of $O(n^3(\log \log n / \log n)^{1/2})$ to $T(n) = O(n^3(\log \log n)^2 / \log n)$. We extend the algorithm to a parallel algorithm for DMM, whose time complexity is $O(\log n)$ and number of processors is $T(n) / \log n$. As an application, we show how to speed up the algorithm for the maximum subarray problem.

1 Introduction

In this paper we consider the all-pairs shortest path (APSP) problem, which computes shortest paths between all pairs of vertices of a directed graph with non-negative real numbers as edge costs. We present an algorithm that computes shortest distances between all pairs of vertices, since shortest paths can be computed as by-products in our algorithm. It is well known that the time complexity of (n, n) -distance matrix multiplication (DMM) is asymptotically equal to that of the APSP problem for a graph with n vertices, as shown in [1, pp 202-206]. Thus we concentrate on DMM in this paper.

Fredman [4] was the first to break the cubic complexity of $O(n^3)$ under RAM, giving $O(n^3(\log \log n / \log n)^{1/3})$. Takaoka [8] improved this complexity to $O(n^3(\log \log n / \log n)^{1/2})$. In this paper we improve the complexity further to $O(n^3(\log \log n)^2 / \log n)$. The above mentioned complexities are all in the worst case. If we go to the average case, we can solve the APSP problem in $O(n^2 \log n)$ time [7]. If edge costs are small integers, the complexity becomes more subcubic, i.e., $O(n^{3-\epsilon})$ for some $\epsilon > 0$, as shown in [9], [2], and [12].

We follow the same framework as those in [4] and [8]. That is, we take a two level divide-and-conquer approach. To multiply the small matrices resulting from dividing the original matrices, we sort distance data, and use the ranks of those data in the sorted lists. As the ranks are small integers, the multiplication can be done efficiently by looking at some precomputed tables.

In Section 2, we describe the two level divide-and-conquer approach. In Section 3, we show how to use ranks to compute DMM. In Section 4, we describe the encoding scheme using the small integers of the ranks. In Section 5, we show how to compute the tables for the efficient multiplication. In Section 6, we show

how to construct DMM by table look-up. In Section 7, we analyze the computing time. In Section 8, we parallelize our algorithm for DMM. Section 9 discusses how to improve algorithms, sequential and parallel, for the maximum subarray problem based on previous sections. Section 10 concludes the paper.

2 Review of Distance Matrix Multiplication

The distance matrix multiplication is to compute the following distance product $C = AB$ for two (n, n) -matrices $A = [a_{ij}]$ and $B = [b_{ij}]$ whose elements are real numbers.

$$c_{ij} = \min_{k=1}^n \{a_{ik} + b_{kj}\}, (i, j = 1, \dots, n) \quad (1)$$

This is also called funny matrix multiplication in some literatures such as [2]. The operation in the right-hand side of (1) is called distance matrix multiplication, and A and B are called distance matrices in this context. This is also viewed as computing n^2 inner products defined for $\mathbf{a} = (a_1, \dots, a_n)$ and $\mathbf{b} = (b_1, \dots, b_n)$ by

$$\mathbf{ab} = \min_{k=1}^n \{a_k + b_k\}$$

Now we divide A , B , and C into (m, m) -submatrices for $N = n/m$ as follows:

$$\begin{pmatrix} A_{1,1} & \dots & A_{1,N} \\ \dots & & \\ A_{N,1} & \dots & A_{N,N} \end{pmatrix} \begin{pmatrix} B_{1,1} & \dots & B_{1,N} \\ \dots & & \\ B_{N,1} & \dots & B_{N,N} \end{pmatrix} = \begin{pmatrix} C_{1,1} & \dots & C_{1,N} \\ \dots & & \\ C_{N,1} & \dots & C_{N,N} \end{pmatrix}$$

Then C can be computed by

$$C_{ij} = \min_{k=1}^N \{A_{ik}B_{kj}\} (i, j = 1, \dots, N), \quad (2)$$

where the product of submatrices is defined similarly to (1) and the “min” operation is defined on submatrices by taking the “min” operation componentwise. Since comparisons and additions of distances are performed in a pair, we omit counting the number of additions for measurement of the complexity. We have N^3 multiplications of distance matrices in (2). Let us assume that each multiplication of (m, m) -submatrices can be done in $T(m)$ computing time, assuming precomputed tables are available. The time for constructing the tables is reasonable when m is small. Then the total time excluding table construction is given by

$$O(n^3/m + (n/m)^3 T(m))$$

In the subsequent sections, we show that $T(m) = O(m^2(m \log m)^{1/2})$. Thus the time becomes $O(n^3(\log m/m)^{1/2})$.

Now we further divide the small (m, m) -submatrices into rectangular matrices in the following way. We rename the matrices A_{ik} and B_{kj} in (2) by A and B . Let $M = m/l$, where $1 \leq l \leq m$. Matrix A is divided into M (m, l) -submatrices A_1, \dots, A_M from left to right, and B is divided into M (l, m) -submatrices $B_1,$

..., B_M from top to bottom. Note that A_k are vertically rectangular and B_k are horizontally rectangular. Then the product $C = AB$ can be given by

$$C = \min_{k=1}^M \{A_k B_k\} \quad (3)$$

The values of m and l will be determined in subsequent sections to achieve the claimed complexity.

3 How to Multiply Rectangular Matrices

We rename again the matrices A_k and B_k in (3) by A and B . In this section we show how to compute AB , that is,

$$\min_{r=1}^l \{a_{ir} + b_{rj}\}, \text{ for } i = 1, \dots, m; j = 1, \dots, m. \quad (4)$$

We assume that the lists of length m

$$(a_{1r} - a_{1s}, \dots, a_{mr} - a_{ms}), (1 \leq r < s \leq l)$$

and

$$(b_{s1} - b_{r1}, \dots, b_{sm} - b_{rm}), (1 \leq r < s \leq l)$$

are already sorted for all r and s such that $1 \leq r < s \leq l$. The time for sorting will be mentioned in Section 7. Let E_{rs} and F_{rs} be the corresponding sorted lists. For each r and s , we merge lists E_{rs} and F_{rs} to form list G_{rs} . Let H_{rs} be the list of ranks of $a_{ir} - a_{is}$ ($i = 1, \dots, m$) in G_{rs} and L_{rs} be the list of ranks of $b_{sj} - b_{rj}$ ($j = 1, \dots, m$) in G_{rs} . Let $H_{rs}[i]$ and $L_{rs}[j]$ be the i th and j th components of H_{rs} and L_{rs} respectively. Then we have

$$G_{rs}[H_{rs}[i]] = a_{ir} - a_{is}, \quad G_{rs}[L_{rs}[j]] = b_{sj} - b_{rj}$$

The lists H_{rs} and L_{rs} for all r and s can be made in $O(l^2 m)$ time, when the sorted lists are available.

We have the following obvious equivalence.

$$a_{ir} + b_{rj} \leq a_{is} + b_{sj} \iff a_{ir} - a_{is} \leq b_{sj} - b_{rj}$$

and

$$a_{ir} - a_{is} \leq b_{sj} - b_{rj} \iff H_{rs}[i] \leq L_{rs}[j]$$

Fredman [4] observed that the information of ordering for all i, j, r , and s in the righthand side of the above formula is sufficient to determine the product AB by a precomputed table. This information is essentially packed in the three dimensional space of $H_{rs}[i]$ ($i = 1, \dots, m; r = 1, \dots, l; s = r+1, \dots, l$), and $L_{rs}[j]$ ($j = 1, \dots, m; r = 1, \dots, l; s = r+1, \dots, l$). We call this the three-dimensional packing.

Takaoka [8] proposed that to compute each (i, j) element of AB , it is enough to know the above ordering for all r and s . We call this the two-dimensional packing. Note that the precomputed table must be obtained within the total time requirement. The two-dimensional packing will therefore allow a larger size of m , leading to a speed-up.

In this paper, we show that a one-dimensional packing scheme is possible. To choose the minimum of

$$x_r^0 = a_{ir} + b_{rj}, (r = 1, \dots, l)$$

we go through a tournament with l participants. We assume m and l are a power of 2. Our theory can be generalized into other cases easily. Specifically we compare in the following pairs. We call this the 0th comparison stage.

$$(x_1^0, x_2^0), (x_3^0, x_4^0), \dots, (x_{l-1}^0, x_l^0)$$

We call the minimum in each pair the winner of the pair. Suppose the minima of those pairs are $x_1^1, x_2^1, \dots, x_{l/2}^1$. Each x_j^1 has two possibilities of being x_{2j-1}^0 or x_{2j}^0 . Then we compare in the following pairs. We call this the 1st comparison stage.

$$(x_1^1, x_2^1), (x_3^1, x_4^1), \dots, (x_{l/2-1}^1, x_{l/2}^1)$$

The winner x_j^2 of (x_{2j-1}^1, x_{2j}^1) has four possibilities of being x_i^0 for $4j-3 \leq i \leq 4j$. If we repeat these stages, we can finish in $\log l - 1$ stages. Comparing in the pair (x_r^0, x_{r+1}^0) , that is, testing “ $x_r^0 \leq x_{r+1}^0$?” is equivalent to comparing in the pair $(H_{r,r+1}[i], L_{r,r+1}[j])$. Thus if we pack two tuples $(H_{12}[i], H_{34}[i], \dots, H_{l-1,l}[i])$ and $(L_{12}[j], L_{34}[j], \dots, L_{l-1,l}[j])$ into single integers, we can know the $l/2$ winners in $O(1)$ time from a precomputed table in the form of an encoded integer. We can take a similar approach for stages 1, 2, ... Thus the time for computing (4) becomes $O(\log l)$. The tables for the k th stage are constructed for all possible remaining $l/2^k$ winners, each of which is one of 2^k participants. The number of necessary tables at the k th stage is given by $(2^k)^{l/2^k}$. See Fig. 1 for $k = 1$. Thus the total number of necessary tables, named by T later, is

$$1 + 2^{l/2} + 4^{l/4} + 8^{l/8} + \dots + (l/2)^2 \leq 2^{l/2} \log l$$

Since the ranks are between 1 and $2m$, the size of each table is bounded by $(2m)^{l/2} (2m)^{l/2} = (2m)^l$.

The time for computing those tables will be mentioned in Section 7. In the following, winners are specified by indices.

Example 1. Let $H_{rs}[i]$ and $L_{rs}[j]$ be given in Figure 1. First we have the two lists

$$\begin{aligned} (H_{1,2}[i], H_{3,4}[i], H_{5,6}[i], H_{7,8}[i]) &= (1, 5, 12, 3) \\ (L_{1,2}[j], L_{3,4}[j], L_{5,6}[j], L_{7,8}[j]) &= (11, 15, 10, 10) \end{aligned}$$

Since $1 < 11, 5 < 15, 12 > 10$, and $3 < 10$, the winners at the first stage are (1, 3, 6, 7). The next lists are thus

$$(H_{1,3}[i], H_{6,7}[i]) = (13, 4), \quad (L_{1,3}[j], L_{6,7}[j]) = (3, 7) \text{ (shown by asterisks)}$$

Since $13 > 3$ and $4 < 7$, the winners at the second stage are (3, 6). The last lists are $(H_{3,6}[i]) = (1)$ and $(L_{3,6}[j]) = (8)$, from which we conclude $a_{i3} + b_{3j}$ is the minimum. All tuples above are encoded in single integers in actual computation. If we had different H and L , we might have other winners such as (2, 4, 5, 8) at the first stage, and (1, 7) at the second stage, etc. All possibilities are shown by asterisk or minus signs for $k = 1$. We have all preparations for those situations in the form of precomputed tables, so each step is done in $O(1)$ time.

H	1	2	3	4	5	6	7	8	L	1	2	3	4	5	6	7	8
	-----									-----							
1		1	13*	5-	6	7	3	4	1		11	3*	2-	7	4	5	6
2			12-	4-	5	8	2	5	2			4-	1-	5	6	7	8
3				5	6	1	10	7	3				15	3	8	1	5
4					2	4	5	1	4					11	3	7	14
5						12	3-	9-	5						10	5-	6-
6							4*	8-	6							7*	9-
7								3	7								10
8									8								

Fig. 1. $H_{rs}[i]$ and $L_{rs}[j]$ for $l = 8$

4 Encoding Lists into Integers

The sequence of k integers x_{k-1}, \dots, x_1, x_0 , where $0 \leq x_i \leq \mu - 1$ for an integer μ , is encoded into an integer by the function h^k such that

$$h^k(x_{k-1}, \dots, x_1, x_0) = x_{k-1}\mu^{k-1} + \dots + x_1\mu + x_0$$

It is obvious that the mapping h^k is a bijection. The function h^k can be computed in $O(k)$ time by Horner's method, assuming that arithmetic operations on integers up to μ^k can be done in $O(1)$ time. We assume later that μ and k are small, so that those operations are done in $O(1)$ time because μ^k can be contained in a single computer word. In the following we omit the superscript k wherever it is clear from the context. We note that the inverse function h^{-1} , that is, decoding, can be computed in $O(k)$ time by successive division. If $h(x_{k-1}, \dots, x_1, x_0) = z$, we define $h^{-1}(z) = (x_{k-1}, \dots, x_1, x_0)$. We express the j th element of $h^{-1}(z)$ by $h^{-1}(z)[j]$, that is, $h^{-1}(z)[j] = x_{k-j}$. We express h by h_1 when we set $\mu = l$ to use h_1 for encoding winners given in indices. We also define mapping h_2 from h by setting $\mu = 2m$. Mapping h_2 is used for encoding ranks H 's and L 's. Let z_1, z_2, \dots, z_r be r winners where r is an even integer. We define mappings f and g for $1 \leq r \leq l$ to determine which ranks to use next after we have those r winners. We omit the subscripts i and j from $H_{rs}[i]$ and $L_{rs}[j]$ for simplicity in the following.

$$f(z_1, z_2, \dots, z_r) = (H_{z_1, z_2}, H_{z_3, z_4}, \dots, H_{z_{r-1}, z_r})$$

$$g(z_1, z_2, \dots, z_r) = (L_{z_1, z_2}, L_{z_3, z_4}, \dots, L_{z_{r-1}, z_r})$$

Those mappings can be computed in $O(r)$ time.

5 How to Compute the Tables

In this section we construct tables T^0, T^1, \dots . The role of table T^k is to determine winners $(x_1, \dots, x_{l/2^{k+1}})$ for the next stage from the k -th stage. Let integers α and β represent encoded forms of $l/2$ ranks in H 's and L 's. The role of z is to

represent winners in an encoded form. At the beginning all are winners. The table T^0 is defined by

$$T^0(z, \alpha, \beta) = h_1(x_1, x_2, \dots, x_{l/2}),$$

where $x_j = 2j - 1$ if $h_2^{-1}(\alpha)[j] < h_2^{-1}(\beta)[j]$, $x_j = 2j$ otherwise, and $z = h_1(1, 2, \dots, l)$. The value of z has just one possibility. Tables T^1, T^2, \dots can be defined similarly using $l/2$ winners, $l/4$ winners, ... as follows:

Let α and β be encoded forms of $l/4$ ranks each and $z = h_1(z_1, \dots, z_{l/2})$ such that $2j - 1 \leq z_j \leq 2j$. The value of z has $2^{l/2}$ possibilities. T^1 is defined by

$$T^1(z, \alpha, \beta) = h_1(x_1, x_2, \dots, x_{l/4}),$$

where $x_j = z_{2j-1}$ if $h_2^{-1}(\alpha)[j] < h_2^{-1}(\beta)[j]$, $x_j = z_{2j}$ otherwise.

Let α and β be encoded forms of $l/8$ ranks and $z = h_1(z_1, \dots, z_{l/4})$ such that $4j - 3 \leq z_j \leq 4j$. The value of z has $4^{l/4}$ possibilities. T^2 is defined by

$$T^2(z, \alpha, \beta) = h_1(x_1, x_2, \dots, x_{l/8}),$$

where $x_j = z_{2j-1}$ if $h_2^{-1}(\alpha)[j] < h_2^{-1}(\beta)[j]$, $x_j = z_{2j}$ otherwise.

Let α and β be two ranks, and $z = h_1(z_1, z_2)$ such that $1 \leq z_1 \leq l/2$ and $l/2 + 1 \leq z_2 \leq l$. The value of z has $(l/2)^2$ possibilities.

$$T^{\log l - 1}(z, \alpha, \beta) = h_1(x) = x,$$

where $x = z_1$ if $\alpha < \beta$, and $x = z_2$ otherwise.

In these definitions, z indexes different tables, and thus subscripting may be more suitable, but we put z in the argument part for notational convenience.

We also define tables R^r and S^r for $0 \leq r \leq l$ by

$$R^r(z) = h_2(f(h_1^{-1}(z))) , \quad S^r(z) = h_2(g(h_1^{-1}(z))),$$

where $h_1^{-1}(z) = (z_1, \dots, z_r)$. Those tables are precomputed for all possible z . The role of R^r and S^r is to determine which ranks to use for the next stage when the r winners are given.

To compute $T^1(z, \alpha, \beta) = h_1(x_1, x_2, \dots, x_{l/4})$, for example, we decode z , α , and β in $O(l)$ time, then test $h_2^{-1}(\alpha)[j] < h_2^{-1}(\beta)[j]$ to get $x_1, \dots, x_{l/4}$, and finally encode $x_1, \dots, x_{l/4}$ spending $O(l)$ time. We do this for all possible z , α , and β . Other tables can be computed similarly in $O(l)$ time. Tables R^r and S^r can also be computed in $O(l)$ time. The total time for R 's and S 's are not greater than that for T 's.

The total time for computing those tables is thus $O(2^{l/2} \log l (2m)^l l)$. Observe

$$O(2^{l/2} \log l (2m)^l l) = O(c^{l \log m}), \text{ for some constant } c > 0.$$

6 Algorithm by Table Look-Up

Using tables T^0, T^1, \dots , we can compute $\min_r \{a_{ir} + b_{rj}\}$ by repeating the following $\log l$ steps. We start from $R^l(z^0) = h_2(H_{1,2}, H_{3,4}, \dots, H_{l-1,l})$ and $S^l(z^0) = h_2(L_{1,2}, L_{3,4}, \dots, L_{l-1,l})$. The last $z^{\log l}$ is the solution index.

$$\begin{aligned}
z^0 &= h_1(1, 2, \dots, l) \text{ (} z^0 \text{ was precomputed)} \\
z^1 &= T^0(z^0, R^l(z^0), S^l(z^0)) \\
z^2 &= T^1(z^1, R^{l/2}(z^1), S^{l/2}(z^1)) \\
&\dots \\
z^{\log l} &= T^{\log l-1}(z^{\log l-1}, R^2(z^{\log l-1}), S^2(z^{\log l-1}))
\end{aligned}$$

Example 2. The 0-th stage in Example 1 is illustrated as follows:

$$\begin{aligned}
z^0 &= h_1(1, 2, 3, 4, 5, 6, 7, 8) \\
z^1 &= T^0(z^0, R^8(z^0), S^8(z^0)) \\
&= T^0(z^0, h_2(1, 5, 12, 3), h_2(11, 15, 10, 10)) \\
&= h_1(1, 3, 6, 7) \\
R^8(z^0) &= h_2(H_{1,2}, H_{3,4}, H_{5,6}, H_{7,8}) = h_2(1, 5, 12, 3), \\
S^8(z^0) &= h_2(L_{1,2}, L_{3,4}, L_{5,6}, L_{7,8}) = h_2(11, 15, 10, 10).
\end{aligned}$$

In the above $h_2(1, 5, 12, 3)$, $h_2(11, 15, 10, 10)$, $h_1(1, 3, 6, 7)$, etc., are to show the contents of tuples. They are actually treated as single integers. The arguments of h_2 are ranks, and those of h_1 are winners.

The time for this algorithm by table look-up is $O(m^2 \log l) = O(m^2 \log m)$. Since there are m/l products $A_k B_k$, we need $O((m/l)m^2 \log m)$ time. To compute minimum component-wise in $\min_{k=1}^{m/l} \{A_k B_k\}$, we need $O((m/l)m^2)$ time. We also need $O(Ml^2 m) = O(lm^2)$ time for Ml^2 mergings as described in Section 3.

We set $l = (m \log m)^{1/2}$ to balance the first and the third of the above three complexities. Then to compute the product of (m, m) -matrices, we take $O(m^2(m \log m)^{1/2})$ time.

7 How to Determine the Size of Submatrices

We determine the size of submatrices in Section 2. Let m be given by

$$m = \log^2 n / (\log^2 c (\log \log n)^3).$$

Then we have $l \leq \log n / (\log c \log \log n)$ for sufficiently large n , and the $O(c^{l \log m})$ time for making the tables can be shown to be $O(n)$. Substituting the value of m for $O(n^3(\log m/m)^{1/2})$, we have the overall computing time for distance matrix multiplication as $O(n^3(\log \log n)^2 / \log n)$.

We note that the time for sorting to obtain the lists E_{rs} , and F_{rs} for all k in Section 3 is $O(Ml^2 m \log m) = O(m^{2.5} \log m)$. This task of sorting, which we call presort, is done for all A_{ij} and B_{ij} in Section 2 in advance, taking

$$O((n/m)^2 m^{2.5} \log m) = O(n^2 \log n / (\log \log n)^{1/2}))$$

time, which is absorbed in the main complexity.

8 Parallel Algorithms

In this section we design a parallel algorithm for DMM, and then mention how to apply the parallel algorithm to the APSP problem. We use a PRAM with EREW memory. For parallel computational models, see [1]. The computation of DMM consists of two parts.

- (I) Computation of n^2 component-wise minima of N matrices.
- (II) Computation of N^3 products $A_{ik}B_{kj}$, $(i, j, k = 1, \dots, N)$.

It is well known that there are PRAM-EREW algorithms for finding the minimum of N numbers and broadcasting a datum to N locations with $O(N/\log N)$ processors and $O(\log N)$ time. See [6], for example for these algorithms. Let us discuss a parallel algorithm for (I) first. Applying the above with $N = n/m$, we have a parallel algorithm for (I) with $O(n^3(\log \log n)^3/\log^3 n)$ processors and $O(\log n)$ time, since n^2 minima can be computed in parallel.

Now we design a parallel algorithm for (II). In the computation of $A_k B_k$, there are $O(l^2)$ independent mergings of lists of length m . Such computation is done for $k = 1, \dots, m/l$ in parallel. There are also $M - 1$ “min” operations on (m, m) -matrices. There is a parallel algorithm for those operations with $O(m^2 l / \log m) = O(m^2(m \log m)^{1/2} / \log n)$ processors and $O(\log m)$ time. The tasks of encoding and table look-up computation can be done by a parallel algorithm with $P = O(m^2(m \log m)^{1/2} / \log m)$ and $T = O(\log m)$. Since N^3 products in (II) can be computed in parallel, we have a parallel algorithm for (II) with P processors and T time, where $m = \log^2 n / (\log \log n)^3$, and

$$P = O((n/m)^3 m^2 (m \log m)^{1/2} / \log m) = O(n^3 \log \log n / \log n)$$

$$T = O(\log m) = O(\log \log n)$$

Since the processor phase with a larger number of processors can be simulated by a smaller number of processors, we can say there is a parallel algorithm for DMM with

$$P = O(n^3 (\log \log n)^2 / (\log n)^2)$$

$$T = O(\log n)$$

$$C = O(n^3 (\log \log n)^2 / \log n)$$

The previous results in [4] and [8] are used in [5] and [10] to establish a parallel algorithm for the APSP problem with subcubic cost and $O(\log^2 n)$ time. The results in this section will improve those results in a similar way.

9 Application to the Maximum Subarray Problem

Now we proceed to the maximum subarray problem for an array of size (n, n) . The cubic algorithm for this problem given by Bentley [3] was improved to subcubic by Tamaki and Tokuyama [11]. We review the simplified subcubic version

in [10]. To ease the description of our algorithm, we give a two-dimensional array $a[1..m, 1..n]$ as input data. The maximum subarray problem is to maximize the array portion $a[k..i, l..j]$, that is, to obtain such indices (k, l) and (i, j) . We suppose the upper-left corner has co-ordinate $(1, 1)$.

Example 3. Let a be given by

-1	2	-3	5	-4	-8	3	-3
2	-4	-6	-8	2	-5	4	1
3	-2	9	-9	3	6	-5	2
1	-3	5	-7	8	-2	2	-6

Then the maximum subarray is given by the rectangle defined by the upper left corner $(3, 5)$ and the lower right corner $(4, 6)$, given by vertical bars.

We assume that $m \leq n$ without loss of generality. We also assume that m and n are powers of 2. We will mention the general case of m and n later. Bentley's algorithm finds the maximum subarray in $O(m^2n)$ time, which is cubic when $m = n$.

The central algorithmic concept in this section is that of prefix sum. The prefix sums $sum[1..n]$ of a one-dimensional array $a[1..n]$ is computed by

```
sum[0] := 0;
for i := 1 to n do sum[i] := sum[i - 1] + a[i];
```

This algorithm can be extended to two dimensions with linear time, the details of which are omitted.

We use distance matrix multiplications of both *min* and *max* versions in this section. We compute the partial sums $s[i, j]$ for array portions of $a[1..i, 1..j]$ for all i and j with boundary condition $s[i, 0] = s[0, j] = 0$. These sums are often called the two-dimensional prefix sums of a . Obviously this can be done in $O(mn)$ time. The outer framework of the algorithm is given below. Note that the prefix sums once computed are used throughout recursion.

Algorithm M: Maximum subarray

1. If the array becomes one element, return its value.
2. Otherwise, if $m > n$, rotate the array 90 degrees.
3. Thus we assume $m \leq n$.
4. Let A_{left} be the solution for the left half.
5. Let A_{right} be the solution for the right half.
6. Let A_{column} be the solution for the column-centered problem.
7. Let the solution be the maximum of those three.

Here the column-centered problem is to obtain an array portion that crosses over the central vertical line with maximum sum, and can be solved in the following way.

$$A_{column} = \max_{k=1, l=0, i=1, j=n/2+1}^{i-1, n/2-1, m, n} \{s[i, j] - s[i, l] - s[k, j] + s[k, l]\}.$$

In the above we first fix i and k , and maximize the above by changing l and j . Then the above problem is equivalent to maximizing the following for $i = 1, \dots, m$ and $k = 1, \dots, i - 1$.

$$A_{column}[i, k] = \max_{l=0, j=n/2+1}^{n/2-1, n} \{-s[i, l] + s[k, l] + s[i, j] - s[k, j]\}$$

Let $s^*[i, j] = -s[j, i]$. Then the above problem can further be converted into

$$A_{column}[i, k] = -\min_{l=0}^{n/2-1} \{s[i, l] + s^*[l, k]\} + \max_{j=n/2+1}^n \{s[i, j] + s^*[j, k]\} \quad (5)$$

The first part in the above is distance matrix multiplication of the *min* version and the second part is of the *max* version. Let S_1 and S_2 be matrices whose (i, j) elements are $s[i, j - 1]$ and $s[i, j + n/2]$. For an arbitrary matrix T , let T^* be that obtained by negating and transposing T . Then the above can be computed by multiplying S_1 and S_1^* by the *min* version and taking the lower triangle, multiplying S_2 and S_2^* by the *max* version and taking the lower triangle, and finally subtracting the former from the latter and taking the maximum from the triangle.

For simplicity, we apply the algorithm on a square array of size (n, n) , where n is a power of 2. Then all parameters m and n appearing through recursion in Algorithm M are power of 2, where $m = n$ or $m = n/2$. We observe the algorithm splits the array vertically and then horizontally. We define the work of computing the three A_{column} 's through this recursion of depth 2 to be the work at level 0. The algorithm will split the array horizontally and then vertically through the next recursion of depth 2. We call this level 1.

Now let us analyze the time for the work at level 0. We can multiply $(n, n/2)$ and $(n/2, n)$ matrices by 4 multiplications of size $(n/2, n/2)$, and there are two such multiplications in (5). We measure the time by the number of comparisons, as the rest is proportional to this. Let $M(n)$ be the time for multiplying two $(n/2, n/2)$ matrices. At level 0, we obtain an A_{column} and two A_{column} 's, spending $12M(n)$ comparisons. Thus we have the following recurrence for the total time $T(n)$.

$$T(1) = 0, \quad T(n) = 4T(n/2) + 12M(n).$$

Theorem 1. *Let c be an arbitrary constant such that $c > 0$. Suppose $M(n)$ satisfies the condition $M(n) \geq (4 + c)M(n/2)$. Then the above $T(n)$ satisfies*

$$T(n) \leq 12(1 + 4/c)M(n).$$

See [10] for proof.

Clearly the complexity of $O(n^3(\log \log n)^2 / \log n)$ for $M(n)$ satisfies the condition of the theorem with some constant $c > 0$. Thus the maximum subarray problem can be solved in $O(n^3(\log \log n)^2 / \log n)$ time. Since we take the maximum of several matrices component-wise in our algorithm, we need an extra term of $O(n^2)$ in the recurrence to count the number of operations. This term can be absorbed by slightly increasing the constant 12 in front of $M(n)$.

If we start from an (m, n) array with $m < n$, the recursion proceeds without a 90-degree rotation until we hit $m = n$. This part will increase the complexity by a factor of $\log(n/m)$.

Suppose n is not given by powers of 2. By embedding the array a in the array of size (n', n') such that n' is the next power of 2 and the gap is filled with 0, we can solve the original problem in the complexity of the same order.

Now we design a parallel algorithm for the maximum subarray problem based on the previous sections, using the PRAM-EREW model. We see all the problems of A_{column} defined through Algorithm M can be computed in parallel. We define a region to be a square array portion defined through recursion. At level 0, there is one region, 4 at level 1, 16 at level 2, etc. We prepare a processor for each region, which we call a task processor. The task of the task processor is to organize the computation of the three A_{column} 's defined for each square region similarly to level 0. We can use Algorithm M as a parallel device to transmit the information, that is, the co-ordinates, of the region for each task processor in parallel, that is, through lines 4 and 5. After the transmission is completed, each task processor will copy necessary data from array s by employing m^2 copying processors, where the region is an (m, m) array for some m , and then use the parallel algorithm described in the previous section, mobilizing the necessary number of processors under its control. Note that each element of array s is seen by $O(\log n)$ copying processors. After all the A_{column} are finished, each task processor can report its finding through Algorithm M towards the top level, spending $O(\log n)$ time. The time for transmission is obviously within $O(\log n)$. The time for computing all A_{column} is dominated at the top level, which is $O(\log n)$. Thus the total time is $O(\log n)$.

We analyze the total cost. At the top level, we require $P(n) = O(n^3(\log \log n)^2 / (\log n)^2)$. At the 1st level, $P(n/2)$ processors are used by a task processor, and there are 4 task processors, etc. Thus the total number of processors, $P(n)$, is given by

$$\begin{aligned} & O(n^3(\log \log n)^2 / \log^2 n) + O(4(n/2)^3(\log \log(n/2))^2 / \log^2(n/2)) \\ & \quad + O(16(n/4)^3(\log \log(n/4))^2 / \log^2(n/4)) + \dots \\ & = O(n^3(\log \log n)^2 / \log^2 n) \end{aligned}$$

That is, the total cost $O(P(n) \log n)$ is asymptotically equal to that of the sequential algorithm.

It is well known that the prefix sums for a one-dimensional array of size n can be done in $O(\log n)$ time with $n/\log n$ processors using a PRAM-EREW. See [6] for the algorithm. If we apply this algorithm in two dimensions, we can have prefix sums in $O(\log n)$ time with $O(n^2/\log n)$ processors in the same computational model. These complexities are absorbed in the above main complexities.

10 Concluding Remarks

We showed an asymptotic improvement on the time complexity of the all-pairs shortest path problem, and parallelized it. The results will have consequences for

application areas where DMM or the APSP problem is used. As an example, we showed that the maximum subarray problem can be solved in the same complexity as that of DMM.

There is a room for improvement of the exponent of the factor $\log \log n$ in the complexities being lowered down in future research.

Part of this work was done while the author was on leave at Osaka University.

References

1. Akl, S, The design and analysis of parallel algorithms, Prentice-Hall, 1989.
2. Alon, N, Galil, and Margalit, On the Exponent of the All Pairs Shortest Path Problem, Jour. Comp. Sys. Sci., vol 54, no. 2, pp 255-262, 1997
3. Bentley, J., Programming Pearls - Perspective on Performance, Comm. ACM, 27 (1984) 1087-1092
4. Fredman, M, New bounds on the complexity of the shortest path problem, SIAM Jour. Computing, vol. 5, pp 83-89, 1976
5. Han, Y, Pan, V. Y., and Reif, J. H., Efficient parallel algorithms for computing all pair shortest paths in directed graphs, Algorithmica, vol 17, pp 399-415, 1997
6. Horowitz, E, Sahni, S, Rajasekaran, S, Computer Algorithms, Computer Science Press, 1998.
7. Moffat, A. and T. Takaoka, An all pairs shortest path algorithm with $O(n^2 \log n)$ expected time, SIAM Jour. Computing, (1987)
8. Takaoka, T., A New Upper Bound on the complexity of the all pairs shortest path problem, Info. Proc. Lett., 43 (1992) 195-199
9. Takaoka, T, Subcubic algorithms for the all pairs shortest path problem, Algorithmica, vol. 20, 309-318, 1998
10. Takaoka, T, Subcubic algorithms for the maximum subarray problem, Proc. Computing:Australasian Theory Symposium (CATS 2002), pp 189-198, 2002.
11. Tamaki, H. and T. Tokuyama, Algorithms for the Maximum Subarray Problem Based on Matrix Multiplication, Proceedings of the 9th SODA (Symposium on Discrete Algorithms), (1998) 446-452
12. Zwick, U, All pairs shortest paths in weighted directed graphs - exact and almost exact algorithms, 39th FOCS, pp 310-319, 1998.

Algorithms for the On-Line Quota Traveling Salesman Problem^{*}

Giorgio Ausiello¹, Marc Demange², Luigi Laura¹, and Vangelis Paschos³

¹ Dip. di Informatica e Sistemistica Università di Roma "La Sapienza"
Via Salaria, 113 - 00198 Roma Italy
{ausiello,laura}@dis.uniroma1.it

² ESSEC, département SID.
Avenue Bernard HIRSH, BP 105, 95021 Cergy Pontoise cedex France
demange@essec.fr

³ Université Paris-Dauphine
Place du Maréchal De Lattre de Tassigny, 75775 Paris Cedex 16, France
paschos@lamsade.dauphine.fr

Abstract. The Quota Traveling Salesman Problem is a generalization of the well known Traveling Salesman Problem. The goal of the traveling salesman is, in this case, to reach a given *quota* of the sales, minimizing the amount of time. In this paper we address the on-line version of the problem, where requests are given over time. We present algorithms for various metric spaces, and analyze their performance in the usual framework of the competitive analysis. In particular we present a 2-competitive algorithm that matches the lower bound for general metric spaces. In the case of the half-line metric space, we show that it is helpful not to move at full speed, and this approach is also used to derive the best on-line polynomial time algorithm known so far for the more general On-Line TSP problem (in the homing version).

1 Introduction

Let us imagine that a traveling salesman is not forced to visit all cities in a single tour but in each city he can sell a certain amount of merchandise and his commitment is to reach a given *quota* of sales, by visiting a sufficient number of cities; then he is allowed to return back home. The problem to minimize the amount of time in which the traveling salesman fulfills his commitment is known as the Quota Traveling Salesman Problem (QTSP for short, see [4, 9] for a definition of the problem) and it is also called Quorum-Cast problem in [11]. Such problem can be seen as a special case of the Prize-Collecting Traveling Salesman Problem (PCTSP¹, [6]) in which again the salesman has to fulfill a

^{*} Work of Giorgio Ausiello and Luigi Laura is partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT); and by "Progetto ALINWEB: Algoritmica per Internet e per il Web", MIUR Programmi di Ricerca Scientifica di Rilevante Interesse Nazionale.

¹ Note that some authors use to name PCTSP the special case that we have called QTSP [5, 8].

quota but now nonnegative penalties are associated to the cities and the cost of the salesman tour is the sum of the distance travelled and the penalties for the non visited cities. QTSP corresponds to the case of PCTSP in which all penalties are 0. A special case of QTSP is the case in which the amount of merchandise that can be sold in any city is 1; this case coincides with the so called k -TSP problem, i.e. the problem to find the minimum tour which visits k cities among the given ones. Moreover this problem is related with the k -MST problem, that is the problem to find the minimum tree which spans k nodes in a graph. Clearly, if all weights are equal to 1 and the quota to be achieved is equal to the number of cities, the problem corresponds to the classic TSP.

The QTSP problem and the other problems mentioned above have been thoroughly studied in the past from the point of view of approximation algorithms. In particular, for the k -TSP problem and for the k -MST problem a polynomial time algorithm with an approximation ratio 3 has been shown in [15] while, for k -MST problem, in the non rooted case, an algorithm with ratio 2.5 has been shown in [1]. For the general PCTSP problem a polynomial time algorithm with polylogarithmic performance guarantee has been given in [4, 5].

In this paper we wish to address the on-line version of the QTSP problem, named OL-QTSP. On line versions of other routing problems such as the traveling salesman problem [3], the traveling repairman problem [13, 16, 18], variants of the dial-a-ride problem [2, 13], have been studied in the literature in the recent years. The most updated results regarding these problems can be found in [17]. In the on-line version of QTSP we imagine that requests are given over time in a metric space and a server (the traveling salesman) has to decide which requests to serve and in what order to serve them, without yet knowing the whole sequence of requests, with the aim of fulfilling the quota by traveling the minimum possible amount of time. As it is common in the evaluation of on-line algorithms [14], the performance of the algorithm (cost of serving the requests needed to fulfill the quota), is matched against the performance of an optimum off-line server, that is a traveling salesman that knows all the requests ahead of time and decides which requests to serve and in what order, in order to fulfill the assigned quota. Clearly the off-line server cannot serve a request before its release time. The ratio between the former and the latter is called *competitive ratio* of the on-line algorithm (see [10]).

In the rest of the paper we first provide a formal definition of the problem and we introduce the corresponding notation. Subsequently, in Section 2 we provide the proof that no algorithm for the OL-QTSP problem can achieve a competitive ratio smaller than 2 in a general metric space, and we also give a simple 2-competitive (hence optimal) on-line algorithm. In Section 3 we discuss the case in which the metric space is the halfline. In such case we introduce an algorithm which achieves a competitive ratio $3/2$ by not moving at full speed. We also show a matching lower bound. In Section 4 we apply the same ‘slow is good’ paradigm used in Section 3, to the design of an on-line algorithm for the OL-TSP and in such way we derive the best on-line polynomial algorithm known so far, whose competitive ratio is 2.78. In the Conclusion section we illustrate some possible developments and open problems.

1.1 Statement of the Problem

Let us define the OL-QTSP problem in a formal way. Let M be a general metric space, let us denote with O the origin of M , and let $r_1 \dots r_n$ be an ordered sequence of requests in M , any request r_i being a triple (t_i, p_i, w_i) where t_i is the release time of the request, p_i is a point in the metric space where the request is released and w_i is the weight of the request. When we refer to the non-weighted problem a request is simply a couple (t_i, p_i) . Note that the sequence is ordered in the sense that if $i < j$ then $t_i < t_j$. The OL-QTSP problem is the problem in which an on-line server (traveling salesman) is supposed to serve at least a quota Q out of the overall amount of requests $\sum_{j=1}^n w_j$ and come back to the origin in the minimum amount of time. In the non-weighted version of the problem the quota Q is simply the minimum number of requests to be served. Clearly no request can be served before its release time.

In the following we will denote by OL the On-Line Server, and by OPT the server moved by the optimum off line server (adversary). Besides, for any sequence of requests σ , by $Z^{OL}(\sigma)$ and $Z^*(\sigma)$ we denote, respectively, the completion time of OL and OPT over the sequence σ ; when there are no ambiguities, we write Z^{OL} and Z^* instead of $Z^{OL}(\sigma)$ and $Z^*(\sigma)$.

Furthermore, with $p^{OL}(t)$ and $p^*(t)$ we denote respectively the position of OL and OPT at time t , and with $d(p_i, p_j)$ we represent the distance between the points p_i and p_j in the metric space M .

2 General Metric Spaces

2.1 Lower Bound of the Problem

We first show the following result, that will be extensively used in the rest of the paper and referenced as the *speed constraint*.

Lemma 1. *No ρ -competitive algorithm, in any metric space, at time t can have its server in a position distant from the origin more than $t \cdot (\rho - 1)$.*

Proof. We simply show that any algorithm that violates the constraint has a competitive ratio bigger than ρ . Assume that time t_1 is the first instant in which the above constraint is not respected, i.e. $d(p^{OL}(t_1), O) > t_1 \cdot (\rho - 1)$. Then, in the same time, the adversary releases a set of requests with overall quota Q in the origin, and these are served immediately by the adversary ($Z^* = t_1$). The completion time of the on-line server is not less than $Z^{OL} > t_1 \cdot (\rho - 1) + t_1 = \rho \cdot t_1$, and this means that the competitive ratio is strictly bigger than ρ : $\frac{Z^{OL}}{Z^*} > \frac{\rho \cdot t_1}{t_1} = \rho$. \square

Just as the TSP is a special case of QTSP, it is easy to observe that the OL-QTSP (in the Homing version, see [3]) is a special case of OL-QTSP. From this point of view from [3] we may trivially derive a lower bound of 2 on the competitive ratio for OL-QTSP. We now prove a stronger result that provides a lower bound for the competitiveness of any on-line algorithm for OL-QTSP in a general metric space, even when the quote Q is bounded by a constant.

Theorem 1. *There is no ρ -competitive algorithm for the OL-QTSP on the line with $\rho < 2$, even with a maximum of 3 requests released.*

Proof. Assume the metric space is the real line. The quota Q is equal to 2. Denote by A and B the abscissa points -1 and $+1$ on the real line, respectively. At moment $t = 0$, a request r_- is released on A and a request r_+ is released at B ; then, no request is released until moment $t = 1$.

Observe first that $p^{OL}(1) \in [-1, 1]$, but, from Lemma 1, we know that at time 1 OL cannot be in -1 or 1 so $p^{OL}(1) \in (-1, 1)$. Then, for $x \in [0, 1]$, define $f(x) = \min\{d(A, p^{OL}(1+x)), d(B, p^{OL}(1+x))\}$ and $g(x) = f(x) - x$. Clearly, g is continuous if the server moves in a continuous way. Also, $g(0) = f(0) > 0$ and $g(1) = f(1) - 1 < 0$, since $p^{OL}(2) \in (-2, 2)$. Hence, there exists $\bar{x} \in [0, 1]$ such that $g(\bar{x}) = 0$, i.e., $f(\bar{x}) = \bar{x}$. From now on let x_0 be the minimum such \bar{x} .

Assume first that $f(x_0) = d(B, p^{OL}(1+x_0)) = x_0$ (this means that $p^{OL}(1+x_0) \in \{1-x_0, 1+x_0\}$). Actually, due to Lemma 1, only one of the two possibilities holds and hence: $p^{OL}(1+x_0) = 1-x_0$. In this case, if the third (last) request r_3 is released in $-1+x_0$ at moment $t = 1+x_0$, then the optimum is equal to 2 (request r_- is served at $t = 1$, r_3 at $1+x_0$, and return to the origin is at time 2). On the other hand, since $p^{OL}(1+x_0) = 1-x_0$, then the server can serve either r_+ and r_3 , or r_- and r_3 in time equal to $(1+x_0)+x_0+1+2(1-x_0) = 4$. In both of the above cases, the OL server cannot guarantee a competitive ratio better than 2.

Assume now $f(x_0) = d(A, p^{OL}(1+x_0)) = x_0$. In this case, considering a request r_3 released in $1-x_0$ at moment $t = 1+x_0$, arguments exactly analogous to the previous ones imply a competitive upper bound of 2. The proof of the theorem is now complete. \square

Note that the above lower bound is achieved with $Q = 2$, the total number of requests released is 3 and two requests are released at time 0: only one request was released *on-line*. Obviously, the result of Theorem 1 does not work for the case $Q = 1$ and also not for the case in which $Q = 2$ and only two requests are released. These cases are addressed in Subsection 2.3

2.2 A 2-Competitive Algorithm

The following algorithm is 2-competitive for the OL-QTSP problem in any metric space:

Algorithm 1 (Wait and Go (WaG)) *For any sequence σ already presented to the server, with at least a quota Q available over all the requests in σ , the algorithm computes $Z^*(\sigma)$; at time $t = Z^*(\sigma)$ the algorithm starts an optimal tour that serves the quota and ends back in the origin.*

Since WaG starts its tour at time Z^* , it concludes it at time $t' \leq 2 \cdot Z^*$. So we can state the following:

Theorem 2. *Algorithm WaG is 2-competitive for the OL-QTSP problem in general metric space.*

Clearly the upper bound provided by the algorithm matches the lower bound previously given for general metric spaces and hence no better competitive ratio can be established in the general case.

Note also that, if we deal with the non-weighted problem, where Q is the number of requests, if Q is fixed and it is not part of the input the optimal solution of the QTSP problem can be computed in polynomial time and, therefore, WaG is a polynomial time algorithm. If Q is part of the input, Tsitsiklis [19] proved that, for the off-line problem, the optimal solution can be computed in polynomial time if the underlying metric space is the line (or the halfline), while, in order to compute $Z^*(\sigma)$ for general metric spaces, we have to resort to an approximation algorithm for the QTSP. Hence if such algorithm provides a solution with approximation ratio c we obtain a $2 \cdot c$ competitive algorithm.

2.3 Particular Cases

In Theorem 1 we proved that, provided that $Q = 2$ and a minimum of 3 requests are released, no algorithm can achieve a competitive ratio better than 2. In this section we deal with two particular cases that we obtain when we release the constraint on the quota and the number of requests.

We start with the case when $Q = 1$, and we prove that, for such case, OL-QTSP admits a competitive ratio $3/2$.

The algorithm guaranteeing such a ratio is the following:

Algorithm 2 (Eventually Replan (ER)) *The server waits for the first moment t_0 where a request r is released at distance at most t_0 ; it goes to serve it at full speed; if on the road, another request is released such that the algorithm can serve it and return to the origin sooner than for r , it decides to serve this new request.*

Theorem 3. *ER is $3/2$ -competitive for the OL-QTSP with $Q = 1$.*

Proof. Suppose that an optimal algorithm should serve a request r_1 released at distance x_1 from the origin at moment t_1 ; then, $Z^* = \max\{t_1, x_1\} + x_1$.

Assume first $Z^* = 2x_1$ ($x_1 \geq t_1$). Then the server moved by ER starts before moment $t = x_1$ and returns to the origin (after serving a request) before $t = 3x_1$. So, the competitive ratio is $\rho \leq 3x_1/2x_1 = 3/2$.

Assume now $Z^* = t_1 + x_1$ ($x_1 \leq t_1$). If OL has not yet moved at moment $t = t_1$, then it moves in order to serve r_1 (since $2x_1 \leq 2t_1$). In this case the competitive ratio is $\rho = (t_1 + 2x_1)/(t_1 + x_1) \leq 3/2$ (recall that we deal with case $x_1 \leq t_1$). If the on-line algorithm has started to move before moment $t = x_1$, then $\rho \leq 3x_1/(x_1 + t_1) \leq 3/2$. Finally, assume that OL starts moving at the moment $t = t_0$, where $x_1 < t_0 < t_1$, in order to serve a request r_0 released at distance $x_0 \leq t_0$. At $t = t_1$, $d(O, OL(t_1)) \leq t_1 - t_0$. Independently on the request, the on-line algorithm has decided to serve at $t = t_1$ (this request may or may not be r_0), it decides to continue its way (guaranteeing $3t_0$), or to serve r_1 . Hence, it can return in the origin before $t = \min\{3t_0, t_1 + (t_1 - t_0) + 2x_1\} = \min\{3t_0, 2Z^* - t_0\} \leq (3/2)Z^*$.

So, competitive ratio $3/2$ is always guaranteed. \square

Now we deal with the case when $Q = 2$ and only two requests are released. We show in the sequel that, also in this case, we can achieve competitive ratio $3/2$.

The algorithm achieving this ratio is the following:

Algorithm 3 (Serve and Wait (SaW)) *The server starts at the first moment t_0 such that it can serve a request r_0 before time $2 \cdot t_0$; once arrived in place r_0 is released, it waits until the second request is released; it serves it and it then returns in the origin.*

Theorem 4. *SaW is $3/2$ -competitive for the OL-QTSP with $Q = 2$ and a total of two requests released.*

Proof. Denote by r_1 and r_2 the two requests released and assume that they are served in this order by an optimal algorithm; assume that this algorithm serves r_1 at instant τ_1 , r_2 at instant τ_2 and that it returns in the origin at instant $\tau_3 = Z^*$. We distinguish the two following cases depending on the order followed by the server moved by SaW.

Assume first that OL serves r_1 before r_2 . It starts before moment $t = \tau_1$ (this moment is the latest departure moment for the server) since it is possible to serve r_1 before moment $2 \cdot \tau_1$ and this request is already released at this moment. Since it aims at r_1 , it serves it at moment $t = \tau_1 + d(0, r_1)$. If request r_2 is already released at this moment, then the server can serve both r_1 and r_2 and return to the origin before moment $\tau_1 + d(0, r_1) + (\tau_2 - \tau_1) + (\tau_3 - \tau_2) = \tau_3 + d(0, r_1) \leq 3/2\tau_3 = 3/2Z^*$. Otherwise, it ends its tour at most at moment $\tau_2 = d(r_1, r_2) + (\tau_3 - \tau_2) = \tau_3 + d(r_1, r_2) \leq 3/2\tau_3 = 3/2Z^*$.

Assume now that the server moved by SaW serves r_2 before r_1 . Then it starts before $t = \tau_1$ (latest departure moment) and serves r_2 before $t = \tau_1 + (\tau_3 - \tau_2)$. At this moment, r_1 is already released; hence server serves it and returns to the origin before moment $\tau_1 + (\tau_3 - \tau_2) + (\tau_2 - \tau_1) + d(0, r_1) \leq 3/2\tau_3$.

Therefore, in both of the above cases, competitive ratio achieved is bounded above by $3/2$ and the proof of the proposition is complete. \square

3 OL-QTSP on the Halfline

In this section we consider the case in which the metric space is the half-line. For such case we are able to show a competitiveness lower bound and a matching upper bound. As we said before the off-line version of this problem is solvable in polynomial time [19].

Theorem 5. *For the OL-QTSP problem, when the metric space is the halfline, for any $Q \geq 1$, there are no ρ -competitive algorithms where $\rho < 3/2$.*

Proof. We start the proof for the case in which $Q = 1$.

At time 1 one request in point 1 is released. Due to the speed constraint (Lemma 1), the server moved by the on-line algorithm cannot reach the request before time $t = 2$, otherwise the adversary might release immediately a request

in the origin. So the on-line server cannot be back in the origin before time $t = 3$, while optimal completion time is 2.

For the case in which $Q > 1$, it is sufficient to use the above sequence in which we release Q requests at the same time in one point. \square

We now present a $3/2$ -competitive algorithm that matches the above lower bound for the problem.

Algorithm 4 (SlowWalk (SW)) *The on-line server OL moves from the origin at half speed until the first time t_0 in which it is possible i) to serve a quota Q over the weights of the requests and ii) come back to the origin in no more than $t_0/2$; then it goes back to the origin at full speed.*

Theorem 6. *For the weighted OL-QTSP problem in the halfline, and for any value of the quota Q , the algorithm SW is ρ -competitive with $\rho = 3/2$.*

Proof. Basically, we have to prove that the on-line algorithm proceeds in such a way to conclude its tour not later than time $t_0 + t_0/2$ while the adversary uses at least time t_0 . Let us first observe that if we consider $t_0 = Z^*$ all requests served by the adversary in an optimal solution have to lie between O and $t_0/2$ and have been released before time t_0 . Therefore, at time $t_0 = Z^*$, SW can turn back because both requirements are met and it can conclude its tour by serving all requests on its way to the origin. The overall time it uses is $t_0 + t_0/2 = 3/2Z^*$ \square

4 Application to the General OL-TSP Problem

In the previous section we have observed that the server can gain in competitiveness by not moving too fast. Other results based on the same approach can be found in [17], where the notion of “non zealous” algorithm is defined, as opposed to the notion of “zealous” algorithm [7] that is, intuitively, an algorithm that never remains idle when there is work to do. In this section we consider the different problem of the general On-Line TSP, in the *homing* version (H-OLTSP [3]) that is the version in which the server has to conclude its tour in the origin. By applying the same “slow is good” paradigm we show that, also in the H-OLTSP case, the “non-zealousness” of the server can help him in achieving a better competitiveness ratio (see also [17]).

We consider a metric space M and denote by Δ -TSP the metric TSP. In H-OLTSP each instance of requests denotes the on-line version; each instance of requests is a sequence $(p_i, t_i)_{i \geq 0}$ and the sequence (t_i) increases. Given such an instance, the traveler has to visit every point p_i (after t_i) in the least possible time, given the fact that, at each time, he only knows the already revealed points. The following on-line algorithm, named **Wait and Slowly Move (WaSM)**, uses a polynomial time r -approximation algorithm A for Δ -TSP.

Without loss of generality one can suppose that, for every Δ -TSP instance (p_1, \dots, p_n) we have:

$$\forall t, d(0, p^*(t)) \leq \max_i(d(0, p_i))$$

Algorithm 5 (Wait and Slowly Move (WaSM))

- 1) $B \leftarrow (3 + \sqrt{17})/2$
- 2) as a new request is presented at time t_0 , then:
 - the on-line traveler comes back to the origin;
 - he waits until time $t_0(1 + 1/B)$;
 - he then follows the solution performed by A with speed $v(t)$ such that:
 - if** $d(p^{OL}(t), 0) \geq t/B$ **then** $v(t) = 0$ (ball-constraint)
 - else** $v(t) = 1$;

Note that the ball-constraint means that, at every time t , the traveler is not allowed to be further than t/B from the origin.

Theorem 7. *If Δ -TSP admits a polynomial time r -approximation algorithm, then WaSM is a $(r + c)$ -competitive algorithm for the H -OLTSP, where $c = (1 + \sqrt{17})/4 \simeq 1.2808$.*

Proof. Observe that the value $B = (3 + \sqrt{17})/2$ satisfies $(B - 1)/2 = 1 + 1/B$. Let us first note that, as the on-line traveler is constrained always to remain at a distance at most t/B from the origin (ball-constraint), then, whenever a new request is presented at time t_0 , he can be guaranteed to be at the origin at time $t_0(1 + 1/B)$.

Let then t be the time the last request is presented. At time $t(1 + 1/B)$ the on-line traveler is in the origin and begins to follow the solution provided by A . Let us consider the two cases in which the on-line traveler has to stop after time $t(1 + 1/B)$ (because of the ball-constraint) or not.

Case 1: $\exists t_1 > 0, p^{OL}(t(1 + 1/B) + t_1) = [t(1 + 1/B) + t_1]/B$. Let us denote by t' be the maximum such time, which means that $p^{OL}(t(1 + 1/B) + t') = [t(1 + 1/B) + t']/B$ and that $\forall \tau > t', p^{OL}(t(1 + 1/B) + \tau) = [t(1 + 1/B) + \tau]/B$ (from then on, the on-line traveler does not stop until the end of the tour).

Note that the distance covered between $t(1 + 1/B)$ and $t(1 + 1/B) + t'$ is at least $[t(1 + 1/B) + t']/B$, consequently

$$Z^{OL} \leq t(1 + 1/B) + t' + r \cdot Z^* - [t(1 + 1/B) + t']/B \quad (1)$$

On the other side the furthest request being at distance at least $[t(1 + 1/B) + t']/B$ from the origin, we have:

$$Z^* \geq 2[t(1 + 1/B) + t']/B \quad (2)$$

From relations 1 and 2 we deduce that, in this case, the related competitive ratio is no more than $r + (B - 1)/2$.

Case 2: $\forall t_1 > 0, p^{OL}(t(1 + 1/B) + t_1) < [t(1 + 1/B) + t_1]/B$. Then,

$$Z^{OL} \leq t(1 + 1/B) + rZ^*$$

and

$$Z^* \geq t$$

Consequently a competitive ratio of $(1 + 1/B + r) = (B - 1)/2 + r$ is guaranteed, which concludes the proof. \square

As a consequence of this theorem, if we use Christofides heuristic [12], that provides an approximation factor of $r = 1.5$, we can easily state the following result.

Corollary 1. *H-OLTSP admits a 2.78-competitive polynomial time algorithm.*

This result improves the 3-competitive result for this problem proved in [3].

5 Conclusions

In this paper the Quota Traveling Salesman Problem has been tackled from the on-line point of view. First we have shown that for general metric spaces the problem can be solved by means of a 2-competitive algorithm while no better competitive algorithm is possible. In particular since the lower bound result is proved on the real line no better on-line algorithm can be expected even for such particular case. On the contrary, if we consider the half-line metric space we showed a $3/2$ competitive algorithm and a matching lower bound. A peculiarity of the OL-QTSP problem is that no zealous algorithm can achieve the best competitive performance.

Although for the classical Homing OL-TSP exponential time zealous algorithms have been shown to reach the best possible competitive ratio, we show a non zealous polynomial time algorithm which achieves a 2.78 competitive ratio, outperforming the best known polynomial time algorithm for the same problem.

The problem to find a polynomial time algorithm for OL-QTSP with a good competitive ratio is still open.

References

1. S. Arya and H. Ramesh. A 2.5-factor approximation algorithm for the k -mst problem. *Information Processing Letters*, 65(3):117–118, 1998.
2. N. Ascheuer, S.O. Krumke, and J. Rambau. On-line dial-a-ride problems: Minimizing the completion time. In *Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *LNCS*, pages 639–650, 2000.
3. G. Ausiello, E. Feuerstein, S. Leonardi, L. Stougie, and M. Talamo. Algorithms for the on-line travelling salesman. *Algorithmica*, 29(4):560–581, 1998.
4. B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k -trees and prize-collecting salesmen. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pages 277–283, May 1995.
5. B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. New approximation guarantees for minimum-weight k -trees and prize-collecting salesmen. *SIAM Journal on Computing*, 1999.
6. E. Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.
7. M. Blom, S.O. Krumke, W.E. de Paepe, and L. Stougie. The online-tsp against fair adversaries. *INFORMS Journal on Computing*, 13:138–148, 2001.

8. A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS-2003)*, Cambridge, MA, 2003. IEEE.
9. A. Blum, R. Ravi, and S. Vempala. A constant-factor approximation algorithm for the k -mst problem. In *ACM Symposium on Theory of Computing*, pages 442–448, 1996.
10. A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
11. S.Y. Cheung and A. Kumar. Efficient quorumcast routing algorithms. In *Proceedings of INFOCOM '94*, volume 2, pages 840–855, Toronto, 1994.
12. N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, G.S.I.A. Carnegie Mellon University, 1976.
13. E. Feuerstein and L. Stougie. On-line, single server dial-a-ride problems. *Theoretical Computer Science*, 268(1):91–105, 2001.
14. A. Fiat and G. Woeginger, editors. *Online Algorithms: The State of the Art*, volume 1442 of *LNCS*. Springer, 1998.
15. N. Garg. A 3 factor approximation algorithm for the minimum tree spanning k vertices. In *Proceedings IEEE Foundations of Computer Science*, pages 302–309, 1996.
16. S.O. Krumke, W.E. de Paepe, D. Poensgen, and L. Stougie. News from the online traveling repairman. In *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science*, volume 2136 of *LNCS*, pages 487–499. Springer, 2001.
17. M. Lipmann. *On-Line Routing Problems*. PhD thesis, Technische Universiteit Eindhoven, 2003.
18. A. Regan S. Irani, X. Lu. On-line algorithms for the dynamic traveling repair problem. In *Proceedings of the 13th Symposium on Discrete Algorithms*, pages 517–524, 2002.
19. J.N. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22:263–282, 1992.

On the Orthogonal Drawing of Outerplanar Graphs

Kumiko Nomura, Satoshi Tayu, and Shuichi Ueno

Dept. of Communications and Integrated Systems
Graduate School of Science and Engineering, Tokyo Institute of Technology
Tokyo 152-8552, Japan
`{nomura,tayu,ueno}@lab.ss.titech.ac.jp`

Abstract. In this paper we show that an outerplanar graph G with maximum degree at most 3 has a 2-D orthogonal drawing with no bends if and only if G contains no triangles. We also show that an outerplanar graph G with maximum degree at most 6 has a 3-D orthogonal drawing with no bends if and only if G contains no triangles.

1 Introduction

We consider the problem of generating orthogonal drawings of outerplanar graphs in the plane and space. The problem has obvious applications in the design of 2-D and 3-D VLSI circuits and optoelectronic integrated systems.

Throughout this paper, we consider simple connected graphs G with vertex set $V(G)$ and edge set $E(G)$. We denote by $d_G(v)$ the degree of a vertex v in G , and by $\Delta(G)$ the maximum degree of vertices of G . G is called a k -graph if $\Delta(G) \leq k$. The connectivity of a graph is the minimum number of vertices whose removal results in a disconnected graph or a single vertex graph. A graph is said to be k -connected if the connectivity of the graph is at least k .

It is well-known that every graph can be drawn in the space so that its edges intersect only at their ends. Such a drawing of a graph G is called a 3-D drawing of G . A graph is said to be planar if it can be drawn in the plane so that its edges intersect only at their ends. Such a drawing of a planar graph G is called a 2-D drawing of G .

A 2-D orthogonal drawing of a planar graph G is a 2-D drawing of G such that each edge is drawn by a sequence of contiguous horizontal and vertical line segments. A 3-D orthogonal drawing of a graph G is a 3-D drawing of G such that each edge is drawn by a sequence of contiguous axis-parallel line segments. Notice that a graph G has a 2-D[3-D] orthogonal drawing only if $\Delta(G) \leq 4$ [$\Delta(G) \leq 6$]. An orthogonal drawing with no more than b bends per edge is called a b -bend orthogonal drawing.

Biedl and Kant [2], and Liu, Morgana, and Simeone [7] showed that every planar 4-graph has a 2-bend 2-D orthogonal drawing with the only exception of the octahedron, which has a 3-bend 2-D orthogonal drawing. Moreover, Kant [6] showed that every planar 3-graph has a 1-bend 2-D orthogonal drawing with the

only exception of K_4 . On the other hand, Garg and Tamassia proved that it is NP-complete to decide if a given planar 4-graph has a 0-bend 2-D orthogonal drawing [5]. Battista, Liotta, and Vargiu showed that the problem can be solved in polynomial time for planar 3-graphs and series-parallel graphs [1].

We show in Section 3 that an outerplanar 3-graph G has a 0-bend 2-D orthogonal drawing if and only if G contains no triangle as a subgraph.

Eades, Symvonis, and Whitesides [4], and Papakostas and Tollis [8] showed that every 6-graph has a 3-bend 3-D orthogonal drawing. Moreover, Wood showed that every 5-graph has a 2-bend 3-D orthogonal drawing [10]. On the other hand, Eades, Stirk, and Whitesides proved that it is NP-complete to decide if a given 5-graph has a 0-bend 3-D orthogonal drawing [3].

We show in Section 4 that an outerplanar 6-graph G has a 0-bend 3-D orthogonal drawing if and only if G contains no triangle as a subgraph.

It is interesting to note that a complete bipartite graph $K_{2,3}$, which is a minimal non-outerplanar graph with no triangles, has no 0-bend 2-D or 3-D orthogonal drawing.

2 Preliminaries

A 2-D drawing of a planar graph G is regarded as a graph isomorphic to G , and referred to as a plane graph. A plane graph partitions the rest of the plane into connected regions. A face is a closure of such a region. The unbounded region is referred to as the external face. We denote the boundary of a face f of a plane graph Γ by $b(f)$. If Γ is 2-connected then $b(f)$ is a cycle of Γ .

Given a plane graph Γ , we can define another graph Γ^* as follows: corresponding to each face f of Γ there is a vertex f^* of Γ^* , and corresponding to each edge e of Γ there is an edge e^* of Γ^* ; two vertices f^* and g^* are joined by the edge e^* in Γ^* if and only if the edge e in Γ lies on the common boundary of faces f and g of Γ . Γ^* is called the (geometric-)dual of Γ .

A graph is said to be outerplanar if it has a 2-D drawing such that every vertex lies on the boundary of the external face. Such a drawing of an outerplanar graph is said to be outerplane. Let Γ be an outerplane graph with the external face f_o , and $\Gamma^* - f_o^*$ be a graph obtained from Γ^* by deleting the vertex f_o^* together with the edges incident to f_o^* . It is easy to see that if Γ is an outerplane graph then $\Gamma^* - f_o^*$ is a forest. In particular, an outerplane graph Γ is 2-connected if and only if $\Gamma^* - f_o^*$ is a tree.

3 2-D Orthogonal Drawing

An edge of a plane graph Γ which is incident to exactly one vertex of a cycle C and located outside C is called a leg of C . A cycle C of Γ is said to be k -legged if C has exactly k legs.

The planar representation $P(\Gamma)$ of a plane graph Γ is the collection of circular permutations of the edges incident to each vertex. Plane graphs Γ and Γ' are said to be equivalent if $P(\Gamma)$ is isomorphic to $P(\Gamma')$.

The following interesting theorem was proved by Rahman, Naznin, and Nishizeki [9].

Theorem I *A plane 3-graph Γ has an equivalent 0-bend 2-D orthogonal drawing if and only if every k -legged cycle in Γ contains at least $4 - k$ vertices of degree 2 in Γ for any k , $0 \leq k \leq 3$. \square*

We show in this section the following theorem.

Theorem 1. *An outerplanar 3-graph G has a 0-bend 2-D orthogonal drawing if and only if G contains no triangle as a subgraph.*

Proof : The necessity is obvious. We show the sufficiency. Let G be an outerplanar 3-graph with no triangles, and Γ be an outerplane graph isomorphic to G . We show that Γ satisfies the condition of Theorem I.

Lemma 1. *If Γ is 2-connected then the boundary of the external face f_o contains at least 4 vertices of degree 2 in Γ .*

Proof of Lemma 1: If Γ is a cycle then the lemma is obvious. Suppose that Γ has more than one cycle. Since Γ is 2-connected, $\Gamma^* - f_o^*$ is a tree. Since Γ contains no triangles, the boundary of a face of Γ corresponding to a leaf of $\Gamma^* - f_o^*$ contains at least 2 vertices of degree 2 in Γ , which also lie on the boundary of f_o . Since a tree has at least 2 leaves, we obtain the lemma. \square

It is easy to see that every cycle C of Γ is the boundary of the external face of a 2-connected outerplane subgraph of Γ . Thus, by Lemma 1, C contains at least 4 vertices of degree 2 in the subgraph. It follows that if C is a k -legged cycle in Γ then C contains at least $4 - k$ vertices of degree 2 in Γ . This completes the proof of the theorem. \square

It should be noted that there exists an outerplanar 4-graph with no triangles that has no 0-bend 2-D orthogonal drawings. Fig. 1 shows such a graph F . F has a pentagon and five squares. If F has a 0-bend 2-D orthogonal drawing then the pentagon and squares are drawn as rectangles. All the squares must lie outside a rectangle R representing the pentagon. This is impossible, however, since there exists a pair of consecutive squares which lie to the same side of R .

4 3-D Orthogonal Drawing

We show in this section the following theorem.

Theorem 2. *An outerplanar 6-graph G has a 0-bend 3-D orthogonal drawing if and only if G contains no triangle as a subgraph. \square*

The necessity is obvious. We will show the sufficiency in the rest of the section.

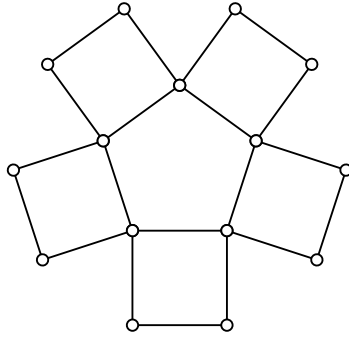


Fig. 1. An outerplanar 4-graph F .

4.1 2-Connected Outerplanar Graphs

We first consider the case when G is 2-connected. Let G be a 2-connected outerplanar 6-graph with no triangles, and Γ be an outerplane graph isomorphic to G . Since Γ is 2-connected, $T^* = \Gamma^* - f_o^*$ is a tree. A leaf r^* of T^* is designated as a root, and T^* is considered as a rooted tree. If g^* is a child of f^* in T^* , f is called the parent face of g , and g is called a child face of f in Γ . The unique edge in $b(f) \cap b(g)$ is called the base of g . The base of r is defined as an edge with both ends of degree 2. Let S^* be a subtree of T^* containing r^* . S^* is also considered as a tree rooted at r^* . $\Gamma(S^*)$ is a subgraph of Γ induced by the vertices on boundaries of faces of Γ corresponding to the vertices of S^* . It should be noted that $\Gamma(S^*)$ is a 2-connected outerplane graph with no triangles. Let f^* be a vertex of S^* , and $f_c^* \in V(T^*) - V(S^*)$ be a child of f^* in T^* . $S^* + f_c^*$ is a rooted tree obtained from S^* by adding f_c^* and an edge (f^*, f_c^*) .

For any face f of Γ , $b(f)$ is a cycle, since Γ is 2-connected. Let $b(f) = \{e_0, e_1, \dots, e_{k-1}\}$, where e_0 is the base of f , and edges e_i and $e_{i+1 \pmod k}$ are adjacent. A 0-bend 2-D orthogonal drawing of f is said to be canonical if f is drawn as a rectangle such that the edges e_2, e_3, \dots , and e_{k-2} are drawn on a side of the rectangle. A drawing of $\Gamma(S^*)$ is said to be canonical if every face is drawn canonically.

Fig.2 shows a rooted tree T^* for F shown in Fig.1, where r is a square face, and a 0-bend 3-D orthogonal canonical drawing of F .

Roughly speaking, we will show that if $\Gamma(S^*)$ has a 0-bend 3-D orthogonal canonical drawing then $\Gamma(S^* + f_c^*)$ also has a 0-bend 3-D orthogonal canonical drawing. The following theorem immediately follows by induction.

Theorem 3. *A 2-connected outerplanar 6-graph with no triangles has a 0-bend 3-D orthogonal drawing.* \square

4.1.1 Proof of Theorem 3 For any $v \in V(\Gamma)$, we define that f_v is a face such that v is on $b(f_v)$ and f_v^* is the nearest vertex to r^* in T^* . We denote by $I_\Gamma(v)$ the set of edges incident with v in Γ .

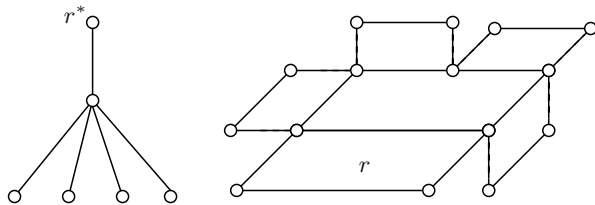


Fig. 2. T^* for F and a 0-bend 3-D orthogonal canonical drawing of F .

Let $\Lambda(S^*)$ be a 0-bend 3-D orthogonal canonical drawing of $\Gamma(S^*)$. We assume without loss of generality that each vertex of $\Lambda(S^*)$ is positioned at a grid-point in the three-dimensional space. Let $\phi : V(\Gamma(S^*)) \rightarrow V(\Lambda(S^*))$ be an isomorphism between $\Gamma(S^*)$ and $\Lambda(S^*)$. The mapping ϕ is called a layout of $\Gamma(S^*)$. If $\phi(v) = (v_x, v_y, v_z)$, we denote $v_x = \phi_x(v)$, $v_y = \phi_y(v)$, and $v_z = \phi_z(v)$. Notice that $\Lambda(S^*)$ is uniquely determined by ϕ .

Let $e_x = (1, 0, 0)$, $e_y = (0, 1, 0)$, $e_z = (0, 0, 1)$ and define that $\mathcal{D} = \{e_x, e_y, e_z, -e_x, -e_y, -e_z\}$. For any $v \in V(\Gamma(S^*))$, β_v is a one-to-one mapping from $I_\Gamma(v)$ to \mathcal{D} . If f is a face with the base $e = (u, v)$, and f_1 is a child face of f with the base $e_1 = (u, v')$, let $\{e_2\} = b(f_1) \cap I_\Gamma(u) - \{e_1\}$. A mapping β_u is said to be admissible if f_1 does not exist or $\beta_u(e_2)$ is orthogonal with both $\beta_u(e)$ and $\beta_u(e_1)$. $\mathcal{B}(S^*) = \{\beta_v \mid v \in V(\Gamma(S^*))\}$ is called a canonical orientation for $\Lambda(S^*)$ if the following conditions are satisfied:

- (B1) For any $v \in V(\Gamma(S^*))$ and $e, e' \in I_\Gamma(v)$, if $e, e' \in b(f)$ for a face $f \neq f_v$ of Γ then $\beta_v(e)$ and $\beta_v(e')$ are orthogonal.
- (B2) If $e_0 = (u, v) \in E(\Gamma(S^*))$ is the base of a face f in Γ , $\{e_1\} = b(f) \cap I_\Gamma(u) - \{e_0\}$, and $\{e_{k-1}\} = b(f) \cap I_\Gamma(v) - \{e_0\}$ then $\beta_u(e_1) = \beta_v(e_{k-1})$.
- (B3) For any $e = (u, v) \in E(\Gamma(S^*))$, $\phi(u) = \phi(v) + m\beta_v(e)$ and $\phi(v) = \phi(u) + m\beta_u(e)$ for some integer m .
- (B4) For each face with base (u, v) , β_u or β_v is admissible.

We prove the theorem by induction. The basis of the induction is stated in the following lemma, whose proof is obvious.

Lemma 2. $\Gamma(r^*)$ has a 0-bend 3-D orthogonal drawing with a canonical orientation. □

Let f^* be a vertex of S^* with a child $f_c^* \in V(T^*) - V(S^*)$.

Lemma 3. If $\Gamma(S^*)$ has a 0-bend 3-D orthogonal canonical drawing with a canonical orientation then $\Gamma(S^* + f_c^*)$ also has a 0-bend 3-D orthogonal canonical drawing with a canonical orientation.

Proof of Lemma 3: Let $\Lambda(S^*)$ be a 0-bend 3-D orthogonal canonical drawing of $\Gamma(S^*)$ with a canonical orientation $\mathcal{B}(S^*) = \{\beta_v \mid v \in V(\Gamma(S^*))\}$, and ϕ be the layout of $\Gamma(S^*)$.

Let $b(f_c) = \{e_0 = (v_0, v_{k-1}), e_1 = (v_0, v_1), \dots, e_{k-1} = (v_{k-2}, v_{k-1})\}$, where e_0 is the base of f_c . We assume without loss of generality that $\beta_{v_0}(e_0) = e_x$

and $\beta_{v_0}(e_1) = e_y$. Now we define a layout ϕ' of $\Gamma(S^* + f_c^*)$. For the vertices v_1, v_2, \dots, v_{k-3} on $b(f_c)$, we define that $\phi'(v_i) = \phi(v_0) + e_y + (i-1)e_x$, $1 \leq i \leq k-3$. We also define that $\phi'(v_0) = \phi(v_0)$, $\phi'_x(v_{k-1}) = \max\{\phi_x(v_{k-1}), \phi_x(v_0) + k - 3\}$, and $\phi'(v_{k-2}) = \phi'(v_{k-1}) + e_y$. Let $l = \phi'_x(v_{k-1}) + k - \phi_x(v_0)$. For each vertex $v \in V(\Gamma(S^*)) - \{v_0, v_{k-1}, v_{k-2}\}$, we define that

$$\begin{aligned}\phi'_x(v) &= \begin{cases} \phi_x(v) & \text{if } \phi_x(v) \leq \phi_x(v_0), \\ \phi_x(v) + l & \text{if } \phi_x(v) > \phi_x(v_0), \end{cases} \\ \phi'_y(v) &= \begin{cases} \phi_y(v) & \text{if } \phi_y(v) \leq \phi_y(v_0), \\ \phi_y(v) + 1 & \text{if } \phi_y(v) > \phi_y(v_0), \end{cases} \\ \phi'_z(v) &= \phi_z(v).\end{aligned}$$

Since $\beta(S^*)$ satisfies (B1), (B2), and (B3), ϕ' is well-defined and induces a 0-bend 3-D orthogonal drawing $\Lambda(S^* + f_c^*)$ of $\Gamma(S^* + f_c^*)$, as easily seen.

It remains to show a canonical orientation $\mathcal{B}(S^* + f_c^*)$ for $\Lambda(S^* + f_c^*)$. Let f_i^* be the children of f_c^* in T^* such that $e_i \in b(f_c)$ is the base of f_i , $1 \leq i \leq k-1$, if any. We first consider a mapping α from the children of f_c^* to $\{0, 1, -1\}$.

Claim 1. *A partial mapping α on $\{f_1, f_{k-1}\}$ with $|\alpha(f_1)| + |\alpha(f_{k-1})| \neq 0$ can be extended to a mapping such that*

- (A1) $\alpha(f_i) \neq \alpha(f_{i+1})$ for $i = 1, k-2$,
- (A2) $|\alpha(f_i) - \alpha(f_{i+1})| = 1$ for $2 \leq i \leq k-3$.

Proof of Claim 1: It suffices to consider the following cases by symmetry.

Case 1 $\alpha(f_1) \leq 0$ and $\alpha(f_{k-1}) = -1$: We define that

$$\alpha(f_i) = \begin{cases} 1 & \text{if } i \text{ is even,} \\ 0 & \text{otherwise.} \end{cases}$$

Case 2 $\alpha(f_1) = 1$ and $\alpha(f_{k-1}) = -1$: We define that

$$\alpha(f_i) = \begin{cases} 0 & \text{if } i \text{ is even,} \\ 1 & \text{otherwise.} \end{cases}$$

It is easy to see that α defined above is a desired mapping. □
 $\mathcal{B}(S^* + f_c^*)$ will be defined as an extension of $\mathcal{B}(S^*)$, that is, $\mathcal{B}(S^*) \subseteq \mathcal{B}(S^* + f_c^*)$. It suffices to define β_{v_i} ($1 \leq i \leq k-2$) for the vertices v_1, v_2, \dots, v_{k-2} on $b(f_c)$. Let $\{e'_1\} = I_\Gamma(v_0) \cap b(f_1) - b(f_c)$ and $\{e'_{k-1}\} = I_\Gamma(v_{k-1}) \cap b(f_{k-1}) - b(f_c)$, if any. We define a partial mapping α as:

$$\alpha(f_1) = \begin{cases} 1 & \text{if } \beta_{v_0}(e'_1) = \beta_{v_0}(e_0) \times \beta_{v_0}(e_1), \\ -1 & \text{if } \beta_{v_0}(e'_1) = -\beta_{v_0}(e_0) \times \beta_{v_0}(e_1), \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\alpha(f_{k-1}) = \begin{cases} 1 & \text{if } \beta_{v_{k-1}}(e'_{k-1}) = \beta_{v_{k-1}}(e_0) \times \beta_{v_{k-1}}(e_{k-1}), \\ -1 & \text{if } \beta_{v_{k-1}}(e'_{k-1}) = -\beta_{v_{k-1}}(e_0) \times \beta_{v_{k-1}}(e_{k-1}), \\ 0 & \text{otherwise,} \end{cases}$$

where \times denotes the exterior product of vectors. Since $\mathcal{B}(S^*)$ satisfies (B4), we have $|\alpha(f_1)| + |\alpha(f_{k-1})| \neq 0$. So let α be a mapping satisfying the conditions of Claim 1.

For each vertex v_i ($1 \leq i \leq k-2$), we label the edges in $I_\Gamma(v_i)$ as follows. Let $e_{v_i}^{(1)} = e_i$. If e is the base of a child face $f_i^{(2)}$ of face f_i , let $e_{v_i}^{(2)} = e$. In general, if e is the base of a child face $f_i^{(j+1)}$ of face $f_i^{(j)}$, let $e_{v_i}^{(j+1)} = e$, if any. If $f_i^{(j)}$ has no such child face and $\{e\} = b(f_i^{(j)}) \cap I_\Gamma(v_i) - \{e_{v_i}^{(j)}\}$, then we defined that $e_{v_i}^{(j+1)} = e$. Let $e_{v_i}^{(6)} = e_{i+1}$. If e is the base of a child face $f_{i+1}^{(5)}$ of face f_{i+1} , let $e_{v_i}^{(5)} = e$. In general, if e is the base of a child face $f_{i+1}^{(j-1)}$ of $f_{i+1}^{(j)}$, let $e_{v_i}^{(j-1)} = e$, if any. If $f_{i+1}^{(j)}$ has no such child face and $\{e\} = b(f_{i+1}^{(j)}) \cap I_\Gamma(v_i) - \{e_{v_i}^{(j)}\}$, then we defined that $e_{v_i}^{(j-1)} = e$.

We first define β_{v_i} for $e_{v_i}^{(1)}$, $e_{v_i}^{(2)}$, $e_{v_i}^{(5)}$, and $e_{v_i}^{(6)}$, if any:

$$\begin{aligned} \beta_{v_i}(e_{v_i}^{(1)}) &= \begin{cases} -e_y & \text{if } i = 1, \\ -e_x & \text{if } 2 \leq i \leq k-2, \end{cases} \\ \beta_{v_i}(e_{v_i}^{(6)}) &= \begin{cases} e_x & \text{if } 1 \leq i \leq k-3, \\ -e_y & \text{if } i = k-2, \end{cases} \\ \beta_{v_i}(e_{v_i}^{(2)}) &= \begin{cases} -e_x & \text{if } \alpha(f_i) = 0 \text{ and } i = 1, \\ e_y & \text{if } \alpha(f_i) = 0 \text{ and } 2 \leq i \leq k-2, \\ e_z & \text{if } \alpha(f_i) = 1, \\ -e_z & \text{if } \alpha(f_i) = -1, \end{cases} \\ \beta_{v_i}(e_{v_i}^{(5)}) &= \begin{cases} e_y & \text{if } \alpha(f_i) = 0 \text{ and } 1 \leq i \leq k-3, \\ e_x & \text{if } \alpha(f_i) = 0 \text{ and } i = k-2, \\ e_z & \text{if } \alpha(f_i) = 1, \\ -e_z & \text{if } \alpha(f_i) = -1. \end{cases} \end{aligned}$$

We next define β_{v_i} for $e_{v_i}^{(3)}$ and $e_{v_i}^{(4)}$, if any. We define that:

$$\begin{aligned} \beta_{v_1}(e_{v_1}^{(3)}) &= \begin{cases} -e_x & \text{if } \alpha(f_1) = \pm 1, \\ e_z & \text{if } \alpha(f_1) = 0 \text{ and } \alpha(f_2) = -1, \\ -e_z & \text{if } \alpha(f_1) = 0 \text{ and } \alpha(f_2) = 1, \end{cases} \\ \beta_{v_1}(e_{v_1}^{(4)}) &= \begin{cases} e_y & \text{if } \alpha(f_2) = \pm 1, \\ e_z & \text{if } \alpha(f_2) = 0 \text{ and } \alpha(f_1) = -1, \\ -e_z & \text{if } \alpha(f_2) = 0 \text{ and } \alpha(f_1) = 1. \end{cases} \end{aligned}$$

if any. It should be noted that $\alpha(f_1) \neq \alpha(f_2)$, since α satisfies (A1). For $2 \leq i \leq k-3$, we define that:

$$\begin{aligned} \beta_{v_i}(e_{v_i}^{(3)}) &= -\beta_{v_i}(e_{v_i}^{(5)}), \\ \beta_{v_i}(e_{v_i}^{(4)}) &= -\beta_{v_i}(e_{v_i}^{(2)}), \end{aligned}$$

if any. We define that:

$$\beta_{v_{k-2}}(e_{v_{k-2}}^{(3)}) = \begin{cases} e_y & \text{if } \alpha(f_{k-2}) = \pm 1, \\ e_z & \text{if } \alpha(f_{k-2}) = 0 \text{ and } \alpha(f_{k-1}) = -1, \\ -e_z & \text{if } \alpha(f_{k-2}) = 0 \text{ and } \alpha(f_{k-1}) = 1, \end{cases}$$

$$\beta_{v_{k-2}}(e_{v_{k-2}}^{(4)}) = \begin{cases} \mathbf{e}_x & \text{if } \alpha(f_{k-1}) = \pm 1, \\ \mathbf{e}_z & \text{if } \alpha(f_{k-1}) = 0 \text{ and } \alpha(f_{k-2}) = -1, \\ -\mathbf{e}_z & \text{if } \alpha(f_{k-1}) = 0 \text{ and } \alpha(f_{k-2}) = 1, \end{cases}$$

if any. It should be noted that $\alpha(f_{k-2}) \neq \alpha(f_{k-1})$, since α satisfies (A1).

Since α satisfies (A1) and (A2), it is not difficult to verify that $\mathcal{B}(S^* + f_c^*)$ defined so far satisfies the conditions (B1) through (B4), and is a canonical orientation for $\Lambda(S^* + f_c^*)$. \square

4.2 General Outerplanar Graphs

We next consider the general case when G is a connected outerplanar 6-graph with no triangles, and complete the proof of Theorem 2.

For graphs G_1 and G_2 , $G_1 \cup G_2$ is a graph defined as $V(G_1 \cup G_2) = V(G_1) \cup V(G_2)$ and $E(G_1 \cup G_2) = E(G_1) \cup E(G_2)$. A subset \mathcal{S} of \mathcal{D} is said to be suitable if vectors in $\mathcal{D} - \mathcal{S}$ can be linearly arranged such that adjacent vectors are orthogonal. Notice that \mathcal{S} is not suitable if and only if $\mathcal{D} - \mathcal{S} = \{\mathbf{e}_a, -\mathbf{e}_a\}$ for some $a \in \{x, y, z\}$. Let Γ be an outerplane graph isomorphic to G , Λ be a 0-bend 3-D orthogonal canonical drawing of Γ with a canonical orientation $\mathcal{B} = \{\beta_v | v \in V(\Gamma)\}$, and $\mathcal{F}_\Gamma(v) = \{\beta_v(e) | e \in I_\Gamma(v)\}$. Λ is said to be suitable if $\mathcal{F}_\Gamma(v)$ is suitable for every vertex $v \in V(\Gamma)$.

In order to complete the proof of Theorem 2, it is sufficient to show the following.

Lemma 4. *Let Γ_1 be an outerplane graph without triangles, and Λ_1 be a suitable 0-bend 3-D orthogonal canonical drawing of Γ_1 with a canonical orientation $\mathcal{B} = \{\beta_v | v \in V(\Gamma_1)\}$ and a layout ϕ^1 . Let Γ_2 be a 2-connected outerplane graph without triangles or a graph consisting of an edge such that $|V(\Gamma_1) \cap V(\Gamma_2)| = 1$ and $\Gamma_1 \cup \Gamma_2$ is a 6-graph. Then, $\Gamma_1 \cup \Gamma_2$ also has a suitable 0-bend 3-D orthogonal canonical drawing.*

Proof of Lemma 4: Let $\{w\} = V(\Gamma_1) \cap V(\Gamma_2)$ and $\mathcal{F}_\Gamma(w) = \{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_s\}$ such that \mathbf{a}_i and \mathbf{a}_{i+1} are orthogonal ($s \leq 6$). We assume without loss of generality that $\mathbf{a}_1 = \mathbf{e}_x$ and $\mathbf{a}_2 = \mathbf{e}_y$. We distinguish two cases.

Case 1 Γ_2 is an edge: Let $E(\Gamma_2) = \{(w, w')\}$. Then, we define a layout ϕ' of $\Gamma_1 \cup \Gamma_2$ as follows:

$$\begin{aligned} \phi'(v) &= \begin{cases} \phi^1(v) & \text{if } \phi_x(v) \leq \phi_x(w), \\ \phi^1(v) + \mathbf{e}_x & \text{if } \phi_x(v) \geq \phi_x(w) + 1, \end{cases} \\ \phi'(w') &= \phi^1(w) + \mathbf{e}_x. \end{aligned}$$

If we define $\beta_w(w, w') = \mathbf{e}_x$, it is easy to see that ϕ' induces a suitable 0-bend 3-D orthogonal canonical drawing of $\Gamma_1 \cup \Gamma_2$ with a canonical orientation.

Case 2 Γ_2 is 2-connected: Let r_2^* be a leaf of T_2^* such that w is on $b(r_2)$, and $b(r_2) = \{(w, v_1), (v_1, v_2), \dots, (v_{k-2}, v_{k-1}), (v_{k-1}, w)\}$. Let e be the base of the unique child face of r_2 . We show a layout of $\Gamma_1 \cup \Gamma_2$ for the case of $e = (w, v_1)$.

Layouts of $\Gamma_1 \cup \Gamma_2$ for other cases can be obtained by similar ways. We define a layout ϕ' of $\Gamma_1 \cup \Gamma_2(r_2^*)$ as follows: for each $u \in V(\Gamma_1)$,

$$\begin{aligned}\phi'_x(u) &= \begin{cases} \phi_x^1(u) & \text{if } \phi_x^1(u) \leq \phi_x^1(w), \\ \phi_x^1(u) + k - 3 & \text{if } \phi_x^1(u) \geq \phi_x^1(w) + 1, \end{cases} \\ \phi'_y(u) &= \begin{cases} \phi_y^1(u) & \text{if } \phi_y^1(u) \leq \phi_y^1(w), \\ \phi_y^1(u) + 1 & \text{if } \phi_y^1(u) \geq \phi_y^1(w) + 1, \end{cases} \\ \phi'_z(u) &= \phi_z^1(u),\end{aligned}$$

and, for each $v_i \in V(\Gamma_2(r_2^*))$,

$$\phi'(v_i) = \begin{cases} \phi^1(w) + \mathbf{e}_y + (i-1)\mathbf{e}_x & 1 \leq i \leq k-2, \\ \phi^1(w) + (k-3)\mathbf{e}_x & i = k-1. \end{cases}$$

If we define $\beta_w(e) = \mathbf{e}_x$ and $\beta_w(w, v_1) = \mathbf{e}_y$, it is easy to verify that ϕ' induces a 0-bend 3-D orthogonal canonical drawing $\Lambda_1 \cup \Lambda(r_2^*)$ of $\Gamma_1 \cup \Gamma_2(r_2^*)$ with a canonical orientation.

Since $\mathcal{F}_{\Lambda_1 \cup \Lambda(r_2^*)}(w)$ is suitable for $\Lambda_1 \cup \Lambda(r_2^*)$, we can produce a suitable 0-bend 3-D orthogonal canonical drawing of $\Gamma_1 \cup \Gamma_2$ with a canonical orientation by Lemma 3. \square

Since it takes $O(n)$ time to draw a face, and the number of faces is $O(n)$ for an n -vertex outerplanar graph, a 0-bend 3-D orthogonal drawing of an n -vertex outerplanar graph can be obtained in $O(n^2)$ time.

References

1. G. Battista, G. Liotta, and F. Vargiu. Spirality and optimal orthogonal drawings. *SIAM J. Comput.*, 27:1764–1811, 1998.
2. T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. LNCS, 855:24–35, 1994.
3. P. Eades, C. Strik, and S. Whitesides. The techniques of komolgorov and bardzin for three-dimensional orthogonal graph drawings. *Information Processing Letters*, 60:97–103, 1996.
4. P. Eades, A. Symvonis, and S. Whitesides. Three-dimensional orthogonal graph drawing algorithms. *Discrete Applied Mathematics*, 103:55–87, 2000.
5. A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31:601–625, 1995.
6. G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16:4–32, 1996.
7. Y. Liu, A. Morgana, and B. Simeone. A linear algorithm for 2-bend embeddings of planar graphs in the two-dimensional grid. *Discrete Applied Mathematics*, 81:69–91, 1998.
8. A. Papakostas and I. G. Tollis. Algorithm for incremental orthogonal graph drawing in three dimensions. *J. Graph Algorithms and Applications*, 3:81–115, 1999.
9. M.S. Rahman, M. Naznin, and T. Nishizeki. Orthogonal drawing of plane graphs without bends. LNCS 2265, pages 392–406, 2001.
10. D. R. Wood. Optimal three-dimensional orthogonal graph drawing in the general position model. *Theoretical Computer Science*, 299:151–178, 2003.

Canonical Decomposition, Realizer, Schnyder Labeling and Orderly Spanning Trees of Plane Graphs (Extended Abstract)

Kazuyuki Miura, Machiko Azuma, and Takao Nishizeki

Graduate School of Information Sciences
Tohoku University, Sendai 980-8579, Japan
{miura,azuma}@nishizeki.ecei.tohoku.ac.jp
nishi@ecei.tohoku.ac.jp

Abstract. A canonical decomposition, a realizer, a Schnyder labeling and an orderly spanning tree of a plane graph play an important role in straight-line grid drawings, convex grid drawings, floor-plannings, graph encoding, etc. It is known that the triconnectivity is a sufficient condition for their existence, but no necessary and sufficient condition has been known. In this paper, we present a necessary and sufficient condition for their existence, and show that a canonical decomposition, a realizer, a Schnyder labeling, an orderly spanning tree, and an outer triangular convex grid drawing are notions equivalent with each other. We also show that they can be found in linear time whenever a plane graph satisfies the condition.

1 Introduction

Recently automatic aesthetic drawing of graphs has created intense interest due to their broad applications, and as a consequence, a number of drawing methods have come out [3–9, 11–13, 15, 18, 21, 23]. The most typical drawing of a plane graph G is the *straight line drawing* in which all vertices of G are drawn as points and all edges are drawn as straight line segments without any edge-intersection. A straight line drawing of G is called a *grid drawing* if the vertices of G are put on grid points of integer coordinates. A straight line drawing of G is called a *convex drawing* if every face boundary is drawn as a convex polygon [4, 22, 23]. A convex drawing of G is called an *outer triangular convex drawing* if the outer face boundary is drawn as a triangle, as illustrated in Fig. 1.

A canonical decomposition, a realizer, a Schnyder labeling and an orderly spanning tree of a plane graph G play an important role in straight-line drawings, convex grid drawings, floor-plannings, graph encoding, etc. [1, 2, 5–8, 12, 13, 15–18, 21]. It is known that the triconnectivity is a sufficient condition for their existence in G [5, 10, 12, 21], but no necessary and sufficient condition has been known. In this paper, we present a necessary and sufficient condition for their existence, and show that a canonical decomposition, a realizer, a Schnyder

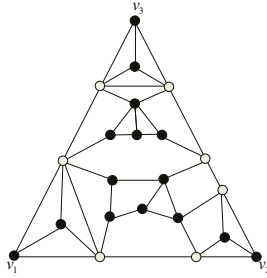


Fig. 1. An outer triangular convex drawing of a plane graph.

labeling, an orderly spanning tree, and an outer triangular convex grid drawing are notions equivalent with each other. We also show that they can be found in linear time whenever G satisfies the condition. Algorithms for finding them have only been available for the triconnected case.

2 Main Theorem and Definitions

Let G be a plane biconnected simple graph. We assume for simplicity that the degrees of all vertices are larger than or equal to three, since the two edges adjacent to a vertex of degree two are often drawn on a straight line. Our main result is the following theorem; some terms will be defined later.

Theorem 1. Let v_1 , v_2 and v_3 be vertices on the outer face $F_o(G)$ of G appearing counterclockwise in this order, as illustrated in Fig. 2. Let P_1 be the path from v_1 to v_2 on $F_o(G)$, let P_2 be the path from v_2 to v_3 , and let P_3 be the path from v_3 to v_1 . Then the following six propositions (a)–(f) are equivalent with each other.

- (a) G has a canonical decomposition with respect to v_1, v_2 and v_3 .
- (b) G has a realizer with respect to v_1, v_2 and v_3 .
- (c) G has a Schnyder labeling with respect to v_1, v_2 and v_3 .
- (d) G has an outer triangular convex grid drawing such that $F_o(G)$ is drawn as a triangle $v_1v_2v_3$.
- (e) G is internally triconnected, and has no separation pair $\{u, v\}$ such that both u and v are on the same path P_i , $1 \leq i \leq 3$. (See Figs. 1,2,3,4.)
- (f) G has an orderly spanning tree such that v_3 is the root, v_1 is the minimum leaf, and v_2 is the maximum leaf.

It is known that (a) \Rightarrow (b), (b) \Leftrightarrow (c) and (c) \Rightarrow (d) [10, 12, 21]. Furthermore, (d) \Rightarrow (e) holds as shown later in Lemma 1. In this paper, we complete a proof of Theorem 1 by proving (e) \Rightarrow (a) in Section 3 and (b) \Leftrightarrow (f) in Section 4.

In the remainder of this section, we present some definitions and known lemmas.

We denote by $G = (V, E)$ an undirected simple graph with vertex set V and edge set E . Let n be the number of vertices of G . An undirected edge joining vertices u and v is denoted by (u, v) . We denote by $\langle u, v \rangle$ a directed edge from u to v .

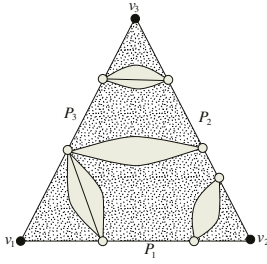


Fig. 2. Illustration for the necessary and sufficient condition.

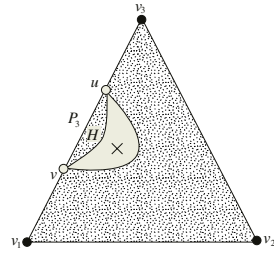


Fig. 3. A separation pair $\{u, v\}$ on P_3 .

A graph is *planar* if it can be embedded in the plane so that no two edges intersect geometrically except at a vertex to which they are both incident. A *plane graph* is a planar graph with a fixed embedding. A plane graph divides the plane into connected regions called *faces*. We denote by $F_o(G)$ the outer face of G . The boundary of $F_o(G)$ is also denoted by $F_o(G)$. A vertex on $F_o(G)$ is called an *outer vertex*, while a vertex not on $F_o(G)$ is called an *inner vertex*. An edge on $F_o(G)$ is called an *outer edge*, while an edge not on $F_o(G)$ is called an *inner edge*.

We call a vertex v of G a *cut vertex* if its removal from G results in a disconnected graph. A graph G is *biconnected* if G has no cut vertex. We call a pair $\{u, v\}$ of vertices in a biconnected graph G a *separation pair* if its removal from G results in a disconnected graph, that is, $G - \{u, v\}$ is not connected. A biconnected graph G is *triconnected* if G has no separation pair. A plane biconnected graph G is *internally triconnected* if, for any separation pair $\{u, v\}$ of G , both u and v are outer vertices and each connected component of $G - \{u, v\}$ contains an outer vertex. In other words, G is internally triconnected if and only if it can be extended to a triconnected graph by adding a vertex in an outer face and joining it to all outer vertices. An internally triconnected plane graph is depicted in Fig. 1, where all the vertices of separation pairs are drawn as white circles. If a biconnected plane graph G is not internally triconnected, then G has a separation pair $\{u, v\}$ as illustrated in Figs. 4(a)–(c) and a “split component” H contains a vertex other than u and v .

We now have the following lemma.

Lemma 1. (d) \Rightarrow (e).

Proof. Suppose that a biconnected plane graph G is not internally triconnected. Then G has a separation pair $\{u, v\}$ as illustrated in Figs. 4(a)–(c), and a “split component” H has a vertex w other than u and v . The degree of w is larger than or equals to three by the assumption of this paper, and hence the two faces marked by \times cannot be simultaneously drawn as convex polygons. Thus G has no outer triangular convex drawing.

Suppose that G has a separation pair $\{u, v\}$ such that both u and v are on P_i , $1 \leq i \leq 3$, as illustrated in Fig. 3. Then G has no outer triangular

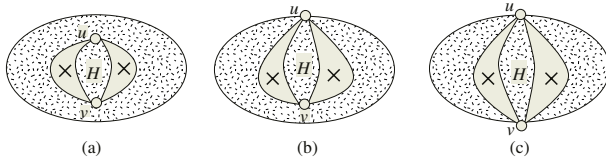


Fig. 4. Biconnected plane graphs which are not internally triconnected.

convex drawing, because P_i must be drawn as a straight line segment, the split component H has a vertex of degree three or more other than u and v , and hence the face marked by \times in Fig. 3 cannot be drawn as a convex polygon.

The two facts above immediately implies (d) \Rightarrow (e). \square

We then define a canonical decomposition [5]. Let $V = \{u_1, u_2, \dots, u_n\}$. Let u_1, u_2 and u_n be three outer vertices appearing counterclockwise on $F_o(G)$ in this order. We may assume that u_1 and u_2 are consecutive on $F_o(G)$; otherwise, let G be the graph obtained by adding a virtual edge (u_1, u_2) to the original graph. Let $\Pi = (V_1, V_2, \dots, V_h)$ be an ordered partition of V into nonempty subsets V_1, V_2, \dots, V_h . Then $V_1 \cup V_2 \cup \dots \cup V_h = V$ and $V_i \cap V_j = \emptyset$ for any indices i and j , $1 \leq i < j \leq h$. We denote by G_k , $1 \leq k \leq h$, the subgraph of G induced by $V_1 \cup V_2 \cup \dots \cup V_k$, and by $\overline{G_k}$ the subgraph of G induced by $V_{k+1} \cup V_{k+2} \cup \dots \cup V_h$. Note that $G = G_h$. We say that Π is a *canonical decomposition* of G (with respect to u_1, u_2 and u_n) if the following three conditions (cd1)–(cd3) hold:

- (cd1) V_1 consists of all the vertices on the boundary of the inner face containing the outer edge (u_1, u_2) , and $V_h = \{u_n\}$.
- (cd2) For each index k , $1 \leq k \leq h$, G_k is internally triconnected.
- (cd3) For each index k , $2 \leq k \leq h$, all the vertices in V_k are outer vertices of G_k , and
 - (a) if $|V_k| = 1$, then the vertex w in V_k has two or more neighbors in G_{k-1} and has at least one neighbor in $\overline{G_k}$ when $k < h$; and
 - (b) if $|V_k| \geq 2$, then the vertices in V_k consecutively appear on $F_o(G_k)$, each of the first and last vertices in V_k has exactly one neighbor in G_{k-1} , and all the other intermediate vertices in V_k have no neighbor in G_{k-1} , and each vertex in V_k has at least one neighbor in $\overline{G_k}$.

A canonical decomposition of the graph in Fig. 1 is illustrated in Fig. 5. Although the definition of a canonical decomposition above is slightly different from one in [5], they are effectively equivalent with each other. The following lemma is known.

Lemma 2. [5] Every triconnected plane graph G has a canonical decomposition.

The definition of a realizer [10] is omitted in this extended abstract, due to the page limitation. The following lemma is known on a realizer and a canonical decomposition.

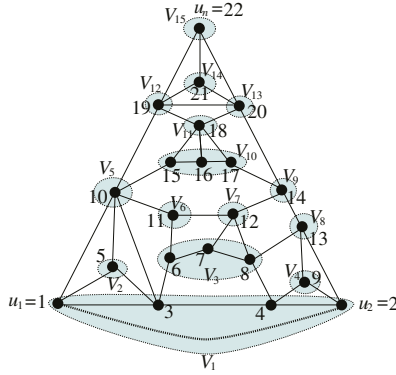


Fig. 5. A canonical decomposition $\Pi = (V_1, V_2, \dots, V_{15})$ of the graph G in Fig. 1.

Lemma 3. [10] If a plane graph G has a canonical decomposition with respect to u_1, u_2 and u_n , then G has a realizer with respect to u_1, u_2 and u_n .

The definition of a Schnyder labeling [12, 21] is omitted in this extended abstract, due to the page limitation. The following three lemmas are known on the Schnyder labeling.

Lemma 4. [12, 21] Every triconnected plane graph G has a Schnyder labeling, and it can be computed in linear time.

Lemma 5. [12] A plane graph G has a realizer if and only if G has a Schnyder labeling.

Lemma 6. [12] If a plane graph G has a Schnyder labeling with respect to a_1, a_2 and a_3 , then G has an outer triangular convex grid drawing such that $F_o(G)$ is drawn as a triangle $a_1a_2a_3$ and the size of grid is $(n-1) \times (n-1)$.

We then define an orderly spanning tree [1, 2, 17]. Let T be a spanning tree of G rooted at an outer vertex u_1 of G . Let u_1, u_2, \dots, u_n be the counterclockwise preordering of vertices in T as illustrated in Fig. 6, where T is drawn by thick lines and each vertex u_i , $1 \leq i \leq n$, is attached an index i . We call the leaf u_α of T having the maximum index the *maximum leaf* of T . Clearly $\alpha = n$. We call the leaf u_β having the minimum index the *minimum leaf* of T . We say that two distinct vertices of G are *unrelated* if any of them is not an ancestor of the other in T . Let $N(u_i)$ be the set of all the neighbors of u_i in G . The set $N(u_i)$ is partitioned into the following four subsets $N_1(u_i)$, $N_2(u_i)$, $N_3(u_i)$ and $N_4(u_i)$:

$$\begin{aligned} N_1(u_i) &= \{u_j \in N(u_i) \mid u_j \text{ is the parent of } u_i \text{ in } T\}, \\ N_2(u_i) &= \{u_j \in N(u_i) \mid j < i, u_j \text{ is not the parent of } u_i\}, \\ N_3(u_i) &= \{u_j \in N(u_i) \mid u_j \text{ is a child of } u_i \text{ in } T\}, \text{ and} \\ N_4(u_i) &= \{u_j \in N(u_i) \mid j > i, u_j \text{ is not a child of } u_i \text{ in } T\}. \end{aligned}$$

Note that $N_1(u_1) = \emptyset$, $|N_1(u_i)| = 1$ for each vertex u_i , $2 \leq i \leq n$, and $|N_3(u_i)| = \emptyset$ for each leaf u_i of T . We call T an *orderly spanning tree* of G if the following conditions (ost1) and (ost2) hold:

- (ost1) For each vertex u_i , $1 \leq i \leq n$, u_i and any vertex $u_j \in N_2(u_i) \cup N_4(u_i)$ are unrelated, and the vertices in $N_1(u_i)$, $N_2(u_i)$, $N_3(u_i)$ and $N_4(u_i)$ appear around u_i counterclockwise in this order, as illustrated in Fig. 7; and
- (ost2) For each leaf u_i of T other than u_α and u_β , $N_2(u_i), N_4(u_i) \neq \emptyset$.

Chiang *et al.* [2] define an orderly spanning tree only for maximal plane graphs, and there is no Condition (ost2) in their definition. They show that a maximal plane graph G has an orderly spanning tree if and only if G has a realizer [2]. We add Condition (ost2) since G is not necessarily a maximal plane graph in this paper.

We have (a) \Rightarrow (b) by Lemma 3, (b) \Leftrightarrow (c) by Lemma 5, (c) \Rightarrow (d) by Lemma 6, and (d) \Rightarrow (e) by Lemma 1. In order to prove Theorem 1, it suffices to prove (e) \Rightarrow (a) and (b) \Leftrightarrow (f).

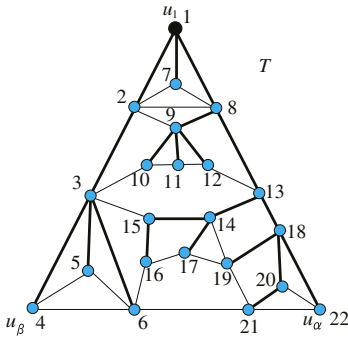


Fig. 6. An orderly spanning tree of the graph G in Fig. 1.

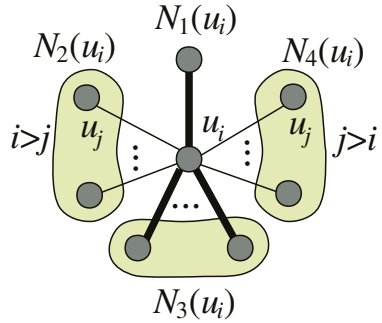


Fig. 7. Four sets $N_1(u_i)$, $N_2(u_i)$, $N_3(u_i)$ and $N_4(u_i)$.

3 Proof of (e) \Rightarrow (a)

In this section, we give a proof of (e) \Rightarrow (a). We first define some terms.

If an internally triconnected plane graph G is not triconnected, then G has a separation pair of outer vertices and hence has a “chord path” defined below when G is not a single cycle.

Let G be a biconnected plane graph, and let w_1, w_2, \dots, w_t be the vertices appearing on $F_o(G)$ clockwise in this order. We call a path Q in G a *chord-path* if Q satisfies the following (i)–(iv):

- (i) Q connects two outer vertices w_p and w_q , $p < q$;
- (ii) $\{w_p, w_q\}$ is a separation pair of G ;
- (iii) Q lies on an inner face; and

- (iv) Q does not pass through any outer edge and any outer vertex other than the ends w_p and w_q .

A chord-path Q connecting w_p and w_q is *minimal* if none of $w_{p+1}, w_{p+2}, \dots, w_{q-1}$ is an end of a chord-path. Thus the definition of a minimal chord-path depends on which vertex is considered as the starting vertex w_1 of $F_o(G)$.

Let $\{x_1, x_2, \dots, x_p\}$, $p \geq 3$, be a set of three or more outer vertices consecutive on $F_o(G)$ such that $d(x_1) \geq 3$, $d(x_2) = d(x_3) = \dots = d(x_{p-1}) = 2$, and $d(x_p) \geq 3$. Then we call the set $\{x_2, x_3, \dots, x_{p-1}\}$ an *outer chain* of G .

We are now ready to prove (e) \Rightarrow (a).

[Proof of (e) \Rightarrow (a)]. Assume that the proposition (e) holds for G , that is, G is internally triconnected and has no separation pair $\{u, v\}$ such that both u and v are on the same path P_i , $1 \leq i \leq 3$, where P_1 connects v_1 and v_2 , P_2 connects v_2 and v_3 , and P_3 connects v_3 and v_1 . We shall show that G has a canonical decomposition $\Pi = (V_1, V_2, \dots, V_h)$. If the outer vertices v_1 and v_2 are not consecutive on $F_o(G)$, then add a virtual edge (v_1, v_2) to the original graph and let G be the resulting graph. Take $u_1 = v_1$, $u_2 = v_2$, and $u_n = v_3$. Take as V_1 the set of all the vertices on the boundary of the inner face containing the edge (u_1, u_2) , and take $V_h = \{u_n\}$. (See Fig. 5.) Then $u_n = v_3 \notin V_1$; otherwise, $\{v_1, v_3\}$ would be a separation pair of G on P_3 , a contradiction. Hence Condition (cd1) of a canonical decomposition holds. Since $G_h = G$ is internally triconnected, Condition (cd2) holds for $k = h$. Since $u_n = v_3$ has degree three or more, Condition (cd3) holds for $k = h$. G is internally triconnected, and the outer vertex u_n of G is not contained in any separation pair of G since (v_1, v_2) is an edge of G and $\{v_3, x\}$ is not a separation pair of G for any vertex x on path P_2 or P_3 . Therefore $G_{h-1} = G - u_n$ is also internally triconnected, and hence (cd2) holds for $k = h - 1$. If $V = V_1 \cup V_h$, then simply setting $h = 2$ we can complete a proof. One may thus assume that $V \neq V_1 \cup V_h$ and hence $h \geq 3$. We choose $V_{h-1}, V_{h-2}, \dots, V_2$ in this order and show that (cd2) and (cd3) hold.

Assume as an inductive hypothesis that $h \geq i + 1 \geq 3$ and the sets $V_h, V_{h-1}, \dots, V_{i+1}$ have been appropriately chosen so that

- (1) (cd2) holds for each index $k \geq i$, and
- (2) (cd3) holds for each index $k \geq i + 1$.

We then show that there is a set V_i of outer vertices of G_i such that

- (1) (cd2) holds for the index $k = i - 1$, and
- (2) (cd3) holds for the index $k = i$.

Let w_1, w_2, \dots, w_t be the outer vertices of G_i appearing clockwise on $F_o(G)$ in this order, where $w_1 = v_1$ and $w_t = v_2$. There are the following two cases to consider.

Case 1: G_i is triconnected. Since G_i is triconnected and at least one vertex in V_{i+1} has a neighbor in G_i , there is an outer vertex $w \notin V_1$ of G_i which has a neighbor in $\overline{G_i}$. We choose the singleton set $\{w\}$ as V_i . Since G_i is triconnected and w is an outer vertex of G_i , $G_{i-1} = G_i - w$ is internally triconnected and w has three or more neighbors in G_{i-1} . Thus (cd2) holds for $k = i - 1$, and (cd3) holds for $k = i$.

Case 2: Otherwise. Since $i \geq 2$, G_i is not a single cycle. G_i is internally triconnected, but is not triconnected. Therefore there is a chord-path for $F_o(G_i)$. Let Q be a minimal chord-path, let w_p and w_q be the two ends of Q , and let $p < q$. Then $q \geq p + 2$; otherwise, G_i would not be internally triconnected. We now have the following two subcases.

Subcase 2a: $\{w_{p+1}, w_{p+2}, \dots, w_{q-1}\}$ is an outer chain of G_i . In this case we choose $\{w_{p+1}, w_{p+2}, \dots, w_{q-1}\}$ as V_i . Since V_i is an outer chain and Q is a minimal chord-path, we have $V_i \cap V_1 = \emptyset$. Each of w_{p+1} and w_{q-1} has exactly one neighbor in G_{i-1} , and each vertex $w \in V_i$ has a neighbor in $\overline{G_i}$ because w has degree three or more in G and has degree two in G_i . One can thus know that (cd3) holds for $k = i$.

We next show that (cd2) holds for $k = i - 1$. Assume for a contradiction that G_{i-1} is not internally triconnected. Then G_{i-1} has either a cut vertex v or a separation pair $\{u, v\}$ having one of the three types illustrated in Fig. 4.

Consider first the case where G_{i-1} has a cut vertex v , as illustrated in Fig. 8. Then v must be an outer vertex of G_i and $v \neq w_p, w_q$; otherwise, G_i would not be internally triconnected. The minimal chord-path Q above must pass through the outer vertex v , contrary to Condition (iv) of the definition of a chord-path.

Consider next the case where G_{i-1} has a separation pair $\{u, v\}$ having one of the three types illustrated in Fig. 4. Then $\{u, v\}$ would be a separation pair of G_i having one of the three types illustrated in Fig. 4, and hence G_i would not be internally triconnected, a contradiction.

Subcase 2b: Otherwise. In this case, every vertex in $\{w_{p+1}, w_{p+2}, \dots, w_{q-1}\}$ has degree three or more in G_i ; otherwise, Q would not be minimal. Furthermore, we can show that at least one vertex w in $\{w_{p+1}, w_{p+2}, \dots, w_{q-1}\}$ has a neighbor in $\overline{G_i}$, as follows. Suppose for a contradiction that none of the vertices in $\{w_{p+1}, w_{p+2}, \dots, w_{q-1}\}$ has a neighbor in $\overline{G_i}$. Then $\{w_p, w_q\}$ is a separation pair of G , and w_p, w_{p+1}, \dots, w_q is a path on $F_o(G)$, as illustrated in Fig. 9. Since $i \leq h - 1$, none of w_p, w_{p+1}, \dots, w_q is $v_3 = u_n$. Thus both w_p and w_q are either on path P_2 or on path P_3 , and hence Proposition (e) would not hold for G , a contradiction.

We choose the singleton set $\{w\}$ as V_i for the vertex w above. Then clearly $V_i \cap V_1 = \emptyset$, and (cd3) holds for the index $k = i$. Since w is not an end of a chord-path of $F_o(G_i)$ and G_i is internally triconnected, $G_{i-1} = G_i - w$ is internally triconnected and hence (cd2) holds for the index $k = i - 1$. \square

4 Proof of (b) \Leftrightarrow (f)

In this section, we give a proof of (b) \Leftrightarrow (f).

Sketchy proof of (b) \Leftrightarrow (f). We can prove that if (T_r, T_b, T_g) is a realizer of G then T_g is an orderly spanning tree of G rooted at r_g . The detail is omitted in this extended abstract, due to the page limitation.

From an orderly spanning tree T of G we can construct a realizer (T_r, T_b, T_g) of G . We take $T_g = T$. For each vertex u_i , $1 \leq i \leq n$, we appropriately choose

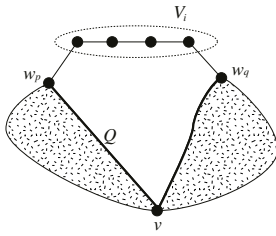


Fig. 8. Graph G_i and the outer chain V_i .

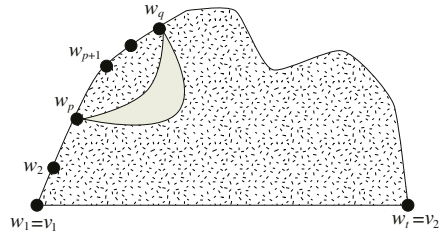


Fig. 9. Graph G_i .

an edge incident to u_i as an outgoing edge of T_b (or T_r) according to some rules. The detail is omitted in this extended abstract, due to the page limitation. \square

Since the proof of (e) \Rightarrow (a) is constructive, we can immediately have a linear algorithm to find a canonical decomposition of G if Proposition (e) holds for G . Moreover, we can examine whether Proposition (e) holds for a given graph G , by using the linear algorithm for decomposing a graph to triconnected components [14]. Furthermore, one can know from the proof of (b) \Rightarrow (f) that if G has a realizer then one can immediately find an orderly spanning tree of G . We thus have the following corollary from Theorem 1.

Corollary 1. If a plane graph G is internally triconnected and has no separation pair $\{u, v\}$ such that both u and v are on the same path P_i , $1 \leq i \leq 3$, then one can find in linear time a canonical decomposition, a realizer, a Schnyder labeling, an orderly spanning tree, and an outer triangular convex grid drawing of G having size $(n-1) \times (n-1)$.

References

1. H. L. Chen, C. C. Liao, H. I. Lu and H. C. Yen, *Some applications of orderly spanning trees in graph drawing*, Proc. Graph Drawing 2002 (GD 2002), LNCS 2528, pp. 332-343 (2002).
2. Y. T. Chiang, C. C. Lin and H. I. Lu, *Orderly spanning trees with applications to graph encoding and graph drawing*, Proc. 12th Annual ACM-SIAM Symp. on Discrete Algorithms, pp. 506-515 (2001).
3. N. Chiba, K. Onoguchi and T. Nishizeki, *Drawing planar graphs nicely*, Acta Inform., 22, pp. 187-201 (1985).
4. N. Chiba, T. Yamanouchi and T. Nishizeki, *Linear algorithms for convex drawings of planar graphs*, in Progress in Graph Theory, J. A. Bondy and U. S. R. Murty (Eds.), Academic Press, pp. 153-173 (1984).
5. M. Chrobak and G. Kant, *Convex grid drawings of 3-connected planar graphs*, International Journal of Computational Geometry and Applications, 7, pp. 211-223 (1997).
6. M. Chrobak and S. Nakano, *Minimum-width grid drawings of plane graphs*, Computational Geometry: Theory and Applications, 10, pp. 29-54 (1998).
7. M. Chrobak and T. Payne, *A linear-time algorithm for drawing planar graphs on a grid*, Information Processing Letters, 54, pp. 241-246 (1995).

8. H. de Fraysseix, J. Pach and R. Pollack, *How to draw a planar graph on a grid*, *Combinatorica*, 10, pp. 41-51 (1990).
9. G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis, *Graph Drawing*, Prentice Hall, NJ (1999).
10. G. Di Battista, R. Tamassia and L. Vismara, *Output-sensitive reporting of disjoint paths*, *Algorithmica*, 4, pp. 302-340 (1999).
11. I. Fáry, *On straight lines representation of plane graphs*, *Acta Sci. Math. Szeged*, 11, pp. 229-233 (1948).
12. S. Felsner, *Convex drawings of planar graphs and the order dimension of 3-polytopes*, *Order*, 18, pp. 19-37 (2001).
13. X. He, *Grid embedding of 4-connected plane graphs*, *Discrete & Computational Geometry*, 17, pp. 339-358 (1997).
14. J. E. Hopcroft and R. E. Tarjan, *Dividing a graph into triconnected components*, *SIAM J. Comput.*, 2, 3, pp. 135-138 (1973).
15. G. Kant, *Drawing planar graphs using the canonical ordering*, *Algorithmica*, 16, pp. 4-32 (1996).
16. G. Kant and X. He, *Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems*, *Theoretical Computer Science*, 172, pp. 175-193 (1997).
17. C. C. Liao, H. I. Lu and H. C. Yen, *Compact floor-planning via orderly spanning trees*, *Journal of Algorithms*, 48, 2, pp. 441-451 (2003).
18. K. Miura, S. Nakano and T. Nishizeki, *Convex grid drawings of four-connected plane graphs*, *Proc. of 11th International Conference ISAAC 2000*, LNCS 1969, pp. 254-265 (2000).
19. S. Nakano, *Planar drawings of plane graphs* Special Issue on Algorithm Engineering IEICE Trans. Inf. Syst., E83-D3 pp. 384-391 (2000).
20. T. Nishizeki and N. Chiba, *Planar Graphs: Theory and Algorithms*, North-Holland, Amsterdam (1988).
21. W. Schnyder, *Embedding planar graphs on the grid*, *Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms*, San Francisco, pp. 138-147 (1990).
22. C. Thomassen, *Plane representations of graphs*, J. A. Bondy, U. S. R. Murty (Eds.), *Progress in Graph Theory*, Academic Press Canada, Don Mills, Ontario, Canada, pp. 43-69 (1984).
23. W. T. Tutte, *How to draw a graph*, *Proc. London Math. Soc.*, 13, pp. 743-768 (1963).

New Bounds on the Number of Edges in a k -Map Graph*

Zhi-Zhong Chen**

Dept. of Math. Sci., Tokyo Denki Univ., Hatoyama, Saitama 350-0394, Japan
chen@r.dendai.ac.jp

Abstract. It is known that for every integer $k \geq 4$, each k -map graph with n vertices has at most $kn - 2k$ edges. Previously, it was open whether this bound is tight or not. We show that this bound is tight for $k = 4$. We also show that this bound is not tight for large enough k ; more precisely, we show that for every $0 < \epsilon < \frac{3}{328}$ and for every integer $k \geq \frac{140}{41\epsilon}$, each k -map graph with n vertices has at most $(\frac{325}{328} + \epsilon)kn - 2k$ edges. We further show that for every positive multiple k of 6, there are infinitely many integers n such that some k -map graph with n vertices has at least $(\frac{11}{12}k + \frac{1}{3})n$ edges.

1 Introduction

Chen, Grigni, and Papadimitriou [2] studied a modified notion of planarity, in which two nations of a (political) map are considered adjacent when they share any *point* of their boundaries (not necessarily an *edge*, as planarity requires). Such adjacencies define a *map graph* (see [3] for a comprehensive survey of known results on map graphs). The map graph is called a *k -map graph* for some positive integer k , if no more than k nations on the map meet at a point. As observed in [2], planar graphs are exactly 3-map graphs, and the adjacency graph of the United States is nonplanar but is a 4-map graph.

The above definitions of map graphs and k -map graphs may be not rigorous enough. For this reason, we give a different rigorous definition. Consider a bipartite graph $B = (V, P; E_B)$ with vertex sets V, P and edge set $E_B \subseteq V \times P$. The *half-square* of B is the simple graph G with vertex set V and edge set $E = \{\{v_1, v_2\} \mid v_1 \in V, v_2 \in V, \text{ and they have a common neighbor in } B\}$. A graph is a *map graph* if it is the half square of a bipartite planar graph. For a positive integer k , a graph is a *k -map graph* if it is the half square of a bipartite planar graph $B = (V, P; E_B)$ in which each $p \in P$ is adjacent to at most k vertices in V .

Chen [1] proved that for every integer $k \geq 4$, each k -map graph with $n \geq k$ vertices has at most $kn - 2k$ edges. It is natural to ask whether this bound is tight or not. Indeed, this question is one of the open questions asked in [3]. In this

* The full version can be found at <http://rnc.r.dendai.ac.jp/~chen/papers/kmap.pdf>

** Supported in part by the Grant-in-Aid for Scientific Research of the Ministry of Education of Japan, under Grant No. 14580390.

paper, we show that this bound is tight when $k = 4$. Moreover, we laboriously show that for each large enough k , this bound is not tight. More precisely, we show that for every $0 < \epsilon < \frac{3}{328}$ and for every integer $k \geq \frac{140}{41\epsilon}$, each k -map graph with $n \geq k$ vertices has at most $(\frac{325}{328} + \epsilon)kn - 2k$ edges. This result has many important consequences. For example, it implies that the arboricity of a k -map graph is at most $(\frac{325}{328} + \epsilon) \cdot k$. Moreover, it implies a linear-time algorithm for coloring a given k -map graph with at most $2 \cdot (\frac{325}{328} + \epsilon) \cdot k$ colors for every $k \geq \frac{140}{41\epsilon}$. Previously, there was no polynomial-time algorithm for coloring a given k -map graph with less than $2k$ colors, although Sanders and Zhao [4] showed that each k -map graph can be colored with at most $\lceil \frac{5}{3}k \rceil$ colors. The proof by them does not lead to a polynomial-time algorithm for coloring a given k -map graph G , because their proof requires that G be given together with its embedding in the plane (or sphere) while it is still unknown how to construct a plane (or sphere) embedding of a given k -map graph in polynomial time.

The above new upper bound (namely, $(\frac{325}{328} + \epsilon)kn - 2k$) may look too loose at first glance. Indeed, since each k -map graph can be colored with at most $\lceil \frac{5}{3}k \rceil$ colors [4], one may be tempted to show that each k -map graph with n vertices has at most $\lceil \frac{5}{6}k \rceil n$ edges. A little unexpectedly, we can show that for every positive multiple k of 6, there are infinitely many integers n such that some k -map graph with n vertices has at least $(\frac{11}{12}k + \frac{1}{3})n$ edges.

2 Basic Definitions

Throughout this paper, a graph may have multiple edges but no loops, while a *simple* graph has neither multiple edges nor loops. Our terminology is standard but we review the main concepts and notation. Let G be a graph. A *bridge* of G is an edge of G whose removal increases the number of connected components in G . G is *bridgeless* if it has no bridge. Let v be a vertex in G . The *degree* of v in G , denoted by $d_G(v)$, is the number of edges incident to v . Note that $d_G(v)$ may be larger than the number of neighbors of v in G because of multiple edges. Vertex v is *isolated* in G if $d_G(v) = 0$. For $U \subseteq V(G)$, the *subgraph of G induced by U* is the graph (U, E_U) where E_U consists of all $e \in E$ such that both endpoints of e are in U . A *cycle* of G is a connected subgraph C of G such that each vertex of C is incident to exactly two edges of C . A *path* of G is a simple connected subgraph P of G such that P is not a cycle and each vertex v of P satisfies $d_P(v) \leq 2$. The *length* of a cycle (respectively, path) is the number of edges on it. The *internal vertices* of a path P are those vertices v on P with $d_P(v) = 2$.

A graph is *planar* if it can be embedded in the plane (respectively, sphere) so that any pair of edges can only intersect at their endpoints; a *plane* (respectively, *sphere*) graph is a planar one together with such an embedding. Let \mathcal{G} be a sphere graph. Consider the set of all points of the sphere that lie on no edge of \mathcal{G} . This set consists of a finite number of topologically connected regions; the closure of each such region is a *face* of \mathcal{G} . Let F be a face of \mathcal{G} . We denote by $V(F)$ (respectively, $E(F)$) the set of all vertices (respectively, edges) of G that are contained in F . The *size* of F is $|V(F)|$. Let F_1 and F_2 be two faces of \mathcal{G} . F_1 and

F_2 *touch* if $V(F_1) \cap V(F_2)$ is not empty. F_1 and F_2 *strongly touch* if $E(F_1) \cap E(F_2)$ is not empty. Obviously, if F_1 and F_2 strongly touch, then they touch. However, the reverse is not necessarily true. When F_1 and F_2 strongly touch in \mathcal{G} , *merging* F_1 and F_2 is the operation of modifying \mathcal{G} by deleting all edges in $E(F_1) \cap E(F_2)$.

Fix an integer $k \geq 3$. A k -*face sphere graph* is a bridgeless sphere graph with no face of size larger than k . Let \mathcal{H} be a k -face sphere graph. Let F be a face of \mathcal{H} . F is *small* if $|V(F)| \leq \lceil \frac{k}{2} \rceil$. F is *large* if it is not small. F is *critical* if it is small and strongly touches exactly two faces of \mathcal{H} . F is *dangerous* if it is critical and strongly touches a face of size less than k . We classify critical faces F into three types as follows:

- Type 1: The boundary of F is a cycle. (Comment: The two faces strongly touching F may or may not strongly touch.)
- Type 2: The boundary of F is formed by two vertex-disjoint cycles.
- Type 3: The boundary of F is formed by two edge-disjoint cycles and the two cycles share exactly one vertex.

The *map graph* of \mathcal{H} is the simple graph whose vertices are the vertices of \mathcal{H} and whose edges are those $\{v_1, v_2\}$ such that there is a face F in \mathcal{H} with $\{v_1, v_2\} \subseteq V(F)$.

Lemma 1. *Let k be an integer larger than 2, and let $G = (V, E)$ be a connected k -map graph with at least k vertices. Then, there is a k -face sphere graph $\mathcal{H} = (V, E_{\mathcal{H}})$ whose map graph is a supergraph of G .*

3 A New Upper Bound

In this section, we show the following theorem:

Theorem 1. *Let $0 < \epsilon < \frac{3}{328}$, and let k be an integer not smaller than $\frac{140}{41\epsilon}$. Then, for every k -face sphere graph \mathcal{H} with at least k vertices, the map graph of \mathcal{H} has at most $(\frac{325}{328} + \epsilon)kn - 2k$ edges, where n is the number of vertices in \mathcal{H} .*

We prove Theorem 1 by induction on $n + f$, where f is the number of faces in \mathcal{H} . In the base case, we have $n = k$ and $f = 1$, and hence the map graph of \mathcal{H} has exactly $\frac{k(k-1)}{2}$ edges.

For the induction step, suppose that $n + f > k$. The following lemma deals with two simple cases.

Lemma 2. *The map graph of \mathcal{H} has at most $(\frac{325}{328} + \epsilon)kn - 2k$ edges if at least one of the following conditions holds:*

1. *There are two strongly touching faces F_1 and F_2 in \mathcal{H} such that merging them does not create a face of size larger than k .*
2. *\mathcal{H} has a small face F such that $V(F)$ contains an isolated vertex v .*

By the above two simple cases, we may make the following assumption:

Assumption 1 *Neither condition in Lemma 2 holds.*

By Assumption 1, no two small faces of \mathcal{H} can strongly touch, and hence each critical face strongly touches two large faces in \mathcal{H} . Moreover, each face of \mathcal{H} has size at least 3.

Corollary 1. *No small face strongly touches exactly one face in \mathcal{H} .*

Lemma 3. *Suppose F is a dangerous face of \mathcal{H} . Let F_1 and F_2 be the two faces strongly touching F in \mathcal{H} . Then, we can shrink F and enlarge either one or both of F_1 and F_2 without modifying the other faces (than F , F_1 , and F_2) of \mathcal{H} to obtain a new k -face sphere graph \mathcal{H}' such that*

- \mathcal{H}' has n vertices and f faces,
- both F_1 and F_2 have size k in \mathcal{H}' (and so F is not a dangerous face of \mathcal{H}'),
- each dangerous face of \mathcal{H}' is also a dangerous face of \mathcal{H} , and
- the map graph of \mathcal{H}' has at least as many edges as the map graph of \mathcal{H} .

In Lemma 3, we can indeed prove that Assumption 1 remains to hold after replacing \mathcal{H} therein by \mathcal{H}' . But the proof is unnecessary here, because even when Assumption 1 does not hold after replacing \mathcal{H} therein by \mathcal{H}' , we can apply Lemma 2 to \mathcal{H}' . In more detail, since \mathcal{H}' has the same numbers of vertices and faces as \mathcal{H} does, we can modify the proof of Lemma 2 to prove that the map graph of \mathcal{H}' has at most $(\frac{325}{328} + \epsilon)kn - 2k$ edges. In turn, the map graph of \mathcal{H} has at most $(\frac{325}{328} + \epsilon)kn - 2k$ edges because the map graph of \mathcal{H}' has at least as many edges as the map graph of \mathcal{H} .

So, we may assume that in Lemma 3, Assumption 1 remains to hold after replacing \mathcal{H} therein by \mathcal{H}' . Then, by repeatedly applying Lemma 3, we can make the following assumption:

Assumption 2 \mathcal{H} has no dangerous face.

Lemma 4. *Suppose that a critical face F of \mathcal{H} has a vertex v_i on its boundary such that v_i has degree more than 2 in \mathcal{H} and is on the boundary of exactly one (say, F_1) of the two faces F_1 and F_2 strongly touching F in \mathcal{H} . Then, we can modify F and F_1 without modifying the other faces (than F and F_1) to obtain a new k -face sphere graph \mathcal{H}' such that*

- \mathcal{H}' has n vertices and f faces,
- F_1 still has size k in \mathcal{H}' while the size of F in \mathcal{H}' is larger than the size of F in \mathcal{H} by 1,
- F strongly touches F_1 , F_2 , and exactly one face $F_3 \notin \{F_1, F_2\}$ in \mathcal{H}' ,
- each critical face of \mathcal{H}' other than F_3 is also a critical face of \mathcal{H} ,
- no face other than F_3 is a dangerous face in \mathcal{H}' , and
- the map graph of \mathcal{H}' has at least as many edges as the map graph of \mathcal{H} .

In Lemma 4, F_3 may be a critical or even dangerous face of \mathcal{H}' . However, this does not cause us any problem. To see this, suppose that F_3 is a critical or dangerous face of \mathcal{H}' . Then, merging F_3 and F in \mathcal{H}' does not yield a face of size larger than k because the size of F_3 (respectively, F) in \mathcal{H}' is at most $\lceil \frac{k}{2} \rceil$

(respectively, $\lceil \frac{k}{2} \rceil + 1$) and hence merging them in \mathcal{H}' yields a face of size at most $\lceil \frac{k}{2} \rceil + (\lceil \frac{k}{2} \rceil + 1) - 2 \leq k$. In turn, since \mathcal{H}' has the same numbers of vertices and faces as \mathcal{H} does, we can modify the proof of Lemma 2 to prove that the map graph of \mathcal{H}' has at most $(\frac{325}{328} + \epsilon)kn - 2k$ edges. Consequently, the map graph of \mathcal{H} has at most $(\frac{325}{328} + \epsilon)kn - 2k$ edges because the map graph of \mathcal{H}' has at least as many edges as the map graph of \mathcal{H} .

So, we may assume that in Lemma 4, F_3 does not become a critical or dangerous face of \mathcal{H}' , and both Assumptions 1 and 2 remain to hold after replacing \mathcal{H} therein by \mathcal{H}' . Then, by repeatedly applying Lemma 4, we can make the following assumption:

Assumption 3 *For every critical face F of \mathcal{H} , each vertex shared by the boundaries of F and exactly one of the two faces strongly touching F in \mathcal{H} has degree 2 in \mathcal{H} .*

Let $k_2 = \lceil \frac{k}{2} \rceil$. Recall that \mathcal{H} has no face of size at most 2. For each j with $3 \leq j \leq k$, let f_j be the number of faces of size j in \mathcal{H} . For each j with $3 \leq j \leq k_2$, let $f_{j,c}$ be the number of critical faces of size j in \mathcal{H} . Let n_0 be the number of isolated vertices in \mathcal{H} .

Lemma 5. $\sum_{j=3}^k (j-2)f_j \leq 2n - n_0 - 4$.

Fix a large constant β independent of k . The choice of β will become clear later.

3.1 The Case Where $\sum_{j=3}^{k_2} (j-2)f_{j,c} \geq \frac{n}{\beta}$

Let \mathcal{H}' be a copy of \mathcal{H} . For each face F of \mathcal{H} , let F' denote the face of \mathcal{H}' corresponding to F . We modify \mathcal{H}' by performing the following steps:

1. For each Type-1 critical face F of \mathcal{H} , if each of P_F and Q_F has at least one internal vertex where P_F and Q_F are the two edge-disjoint paths shared by F and the two faces strongly touching F in \mathcal{H} , then triangulate F' in \mathcal{H}' by adding $|V(F)| - 3$ new edges $\{x, y\}$ such that x is an internal vertex of P_F and y is an internal vertex of Q_F ; otherwise, triangulate F' in \mathcal{H}' arbitrarily.
2. For each Type-2 critical face F of \mathcal{H} , triangulate F' in \mathcal{H}' by adding $|V(F)|$ new edges $\{x, y\}$ such that x is on P_F and y is on Q_F where P_F and Q_F are the two vertex-disjoint cycles forming the boundary of F .
3. For each Type-3 critical face F , triangulate F' in \mathcal{H}' by adding $|V(F)| - 2$ new edges $\{x, y\}$ such that x is on P_F but not on Q_F and y is on Q_F but not on P_F where P_F and Q_F are the two edge-disjoint cycles forming the boundary of F .
4. For each noncritical face F , triangulate F' in \mathcal{H}' by adding at least $(|V(F)| - 3) + i_F$ new edges, where i_F is the number of isolated vertices in F .

How many edges are in the map graph of \mathcal{H} but are not in the map graph of \mathcal{H}' ? To answer this question, we first prove the following lemma:

Lemma 6. *Let F be a face of \mathcal{H} . Then, the following statements hold:*

1. *If F is a Type-1 critical face of \mathcal{H} , then after Step 1 above, the map graph of \mathcal{H} has at most $\frac{(|V(F)|-2)^2}{4} - (|V(F)| - 3)$ edges between vertices in $V(F)$ that are not edges in the map graph of \mathcal{H}' .*

2. *If F is a Type-2 critical face of \mathcal{H} , then after Step 2 above, the map graph of \mathcal{H} has at most $\frac{|V(F)|^2}{4} - |V(F)|$ edges between vertices in $V(F)$ that are not edges in the map graph of \mathcal{H}' .*

3. *If F is a Type-3 critical face of \mathcal{H} , then after Step 3 above, the map graph of \mathcal{H} has at most $\frac{(|V(F)|-2)^2}{4} - (|V(F)| - 2)$ edges between vertices in $V(F)$ that are not edges in the map graph of \mathcal{H}' .*

4. *If F is a noncritical face of \mathcal{H} , then after Step 4 above, the map graph of \mathcal{H} has at most $\frac{(|V(F)|-2)(|V(F)|-3)}{2}$ edges between vertices in $V(F)$ that are not edges in the map graph of \mathcal{H}' .*

Since \mathcal{H}' becomes a triangulated sphere graph, its map graph has at most $3n - 6$ edges. Moreover, $\frac{(|V(F)|-1)^2}{4} - |V(F)| + 3 \geq \frac{|V(F)|^2}{4} - |V(F)|$ if and only if $|V(F)| \leq 6.5$. Thus, by Lemma 6, we have the following corollary.

Corollary 2. *The number of edges in the map graph of \mathcal{H} is at most*

$$3n - 6 + \sum_{j=3}^k \frac{(j-2)(j-3)}{2} (f_j - f_{j,c}) + \sum_{j=3}^6 \left(\frac{(j-1)^2}{4} - j + 3 \right) f_{j,c} + \sum_{j=7}^{k_2} \left(\frac{j^2}{4} - j \right) f_{j,c}.$$

Now, we are ready to show a key lemma.

Lemma 7. *Suppose that $\sum_{j=3}^{k_2} (j-2)f_{j,c} \geq \frac{n}{\beta}$. Then, the map graph of \mathcal{H} has at most $(1 - \frac{3}{8\beta} + \frac{9}{8\beta k})kn - 2k$ edges.*

Proof. Since $k \geq \frac{140}{41\epsilon}$ and $\epsilon < \frac{3}{328}$, we have $k \geq 373$ and $k_2 \geq 186$. In turn, for every $j \in \{3, \dots, 6\}$, $\frac{(j-1)^2}{4} - j + 3 \leq \frac{k_2-2}{4}(j-2)$. Moreover, for every $j \in \{7, 8, \dots, k_2\}$, $\frac{j^2}{4} - j \leq \frac{k_2-2}{4}(j-2)$. Thus, by Corollary 2, the number of edges in the map graph of \mathcal{H} is at most

$$\begin{aligned} & 3n - 6 + \frac{k-3}{2} \sum_{j=3}^k (j-2)(f_j - f_{j,c}) + \frac{k_2-2}{4} \sum_{j=3}^{k_2} (j-2)f_{j,c} \\ & \leq 3n - 6 + \frac{k-3}{2} (2n - 4 - \frac{n}{\beta}) + \frac{k_2-2}{4} \cdot \frac{n}{\beta} \\ & \leq 3n - 6 + \frac{k-3}{2} (2n - 4 - \frac{n}{\beta}) + \frac{k-3}{8} \cdot \frac{n}{\beta} \\ & = (1 - \frac{3}{8\beta} + \frac{9}{8\beta k})kn - 2k. \end{aligned}$$

The first inequality above follows from Lemma 5, the assumption that $\sum_{j=3}^{k_2} (j-2)f_{j,c} \geq \frac{n}{\beta}$, and the observation that $\frac{k-3}{2} \geq \frac{k_2-2}{4}$. The second inequality above follows from the observation that $\frac{k-3}{2} \geq k_2 - 2$. \square

3.2 The Case Where $\sum_{j=3}^{k_2} (j-2)f_{j,c} < \frac{n}{\beta}$

Let \mathcal{H}_c be a copy of \mathcal{H} . We modify \mathcal{H}_c by performing the following steps for each critical face F of \mathcal{H}_c :

1. Find a face F_1 strongly touching F such that the number h_F of vertices shared by F and F_1 only is not smaller than the number ℓ_F of vertices shared by F and F_2 only, where F_2 is the other face strongly touching F in \mathcal{H}_c .
(Comment: Merging F and F_1 yields a face of size exactly $|V(F_1)| + \ell_F$.)
2. Merge F and F_1 into a single face F_3 , and further delete exactly ℓ_F isolated vertices that were shared by F and F_1 only before the merge.
(Comment: We claim that the size of F_3 is k after this step. To see this, first recall that before this step, the size of F_1 is k by Assumption 2. Now, by the comment on Step 1, the claim holds. By the claim, executing the two steps does not create dangerous faces.)

By the above two steps, we have the following lemma immediately.

Lemma 8. *\mathcal{H}_c has no critical face. Moreover, Assumption 1 remains to hold even after replacing \mathcal{H} therein by \mathcal{H}_c .*

The next lemma will be used to show that the number of edges in the map graph of \mathcal{H}_c is not so smaller than the number of edges in the map graph of \mathcal{H} .

Lemma 9. *The number of edges in the map graph of \mathcal{H} is larger than the number of edges in the map graph of \mathcal{H}_c by at most $\frac{9}{8}k(n - n_{\mathcal{R}})$, where $n_{\mathcal{R}}$ is the number of vertices in \mathcal{H}_c .*

A crucial point is that the number of edges in the map graph of \mathcal{H}_c is not so large. We will show this below.

A trouble with \mathcal{H}_c is that some large face in \mathcal{H}_c may have very few edges on its boundary. So, instead of \mathcal{H}_c , we will work on a new k -face sphere graph \mathcal{R} defined as follows. Initially, \mathcal{R} is a copy of \mathcal{H}_c . We then modify this initial \mathcal{R} by performing the following step for each large face F in it:

- If F has at least $\lceil \frac{k_2+2}{10} \rceil$ isolated vertices, then use new edges to connect the isolated vertices in F into a cycle C_F , in such a way that F is split into two faces F' and F'' one of which (say, F') has no vertex in its interior and has C_F as its boundary.
(Comment: F'' contains the same vertices as F did, and hence is large. Moreover, F'' contains no isolated vertex. On the other hand, F' may be a small face, but its boundary must contain at least $\lceil \frac{k_2+2}{10} \rceil$ edges. Moreover, F' strongly touches F'' only.)

By the above comment, the map graph of \mathcal{H}_c is the same as the map graph of \mathcal{R} . So, it suffices to work on \mathcal{R} instead of \mathcal{H}_c .

We classify the small faces of \mathcal{R} into two types as follows. For each small face F of \mathcal{R} , if F is also a small face of \mathcal{H}_c , then we say that F is *old*; otherwise, we say that F is *new*. The following lemma is clear from Lemma 8 and the construction of \mathcal{R} :

Lemma 10. *The following statements hold:*

1. *No two small faces of \mathcal{R} strongly touch in \mathcal{R} .*
2. *Each old small face of \mathcal{R} strongly touches at least three large faces of \mathcal{R} .*
3. *Each new small face of \mathcal{R} strongly touches exactly one large face of \mathcal{R} , and its boundary contains at least $\lceil \frac{k_2+2}{10} \rceil$ edges.*
4. *Each large face of \mathcal{R} has at most $\lceil \frac{k_2+2}{10} \rceil - 1$ isolated vertices. Consequently, the boundary of each large face contains at least $(k_2 + 2) - \lceil \frac{k_2+2}{10} \rceil$ edges.*

Note that the number $n_{\mathcal{R}}$ in Lemma 9 is also the number of vertices in \mathcal{R} . We need several more notations. Let

- $m_{\mathcal{R}}$ be the number of edges in \mathcal{R} ,
- λ_j be the number of faces of size j in \mathcal{R} for each $j \geq 3$,
- λ_{big} be the number of large faces in \mathcal{R} ,
- λ_{new} be the number of new small faces in \mathcal{R} ,
- λ_{old} be the number of old small faces in \mathcal{R} ,
- \mathcal{R}^* be the dual graph of \mathcal{R} ,
- S^* be the underlying simple graph of \mathcal{R}^* (i.e., S^* is the simple graph obtained from G by deleting multiple edges), and
- m_{S^*} be the number of edges in S^* .

Lemma 11. $m_{\mathcal{R}} \geq \frac{k}{40}(9\lambda_{\text{big}} + \lambda_{\text{new}})$ and $m_{S^*} \leq 9\lambda_{\text{big}} + \lambda_{\text{new}}$.

By Lemma 11, the average number of multiple edges between a pair of adjacent vertices in \mathcal{R}^* is at least $\frac{k}{40}$, which is large if k is large. Moreover, if there are many multiple edges between a pair $\{F_1, F_2\}$ of adjacent vertices in \mathcal{R}^* , then the two faces F_1 and F_2 of \mathcal{R} must share many vertices. This is a key for us to show that the map graph of \mathcal{R} does not have so many edges. We will clarify this below.

Let \mathcal{R}' be a copy of \mathcal{R} . For each face F of \mathcal{R} , let F' be the face of \mathcal{R}' corresponding to F . We modify \mathcal{R}' by triangulating each face of \mathcal{R}' . We want to estimate how many edges are in the map graph of \mathcal{R} but are not in the map graph of \mathcal{R}' .

Lemma 12. *If we modify \mathcal{R} by triangulating exactly one face F , then the number of edges in the map graph of \mathcal{R} decreases by at most $\frac{(|V(F)|-2)(|V(F)|-3)}{2}$. Consequently, the number of edges in the map graph of \mathcal{R} is at most $3n_{\mathcal{R}} - 6 + \sum_{j=3}^k \frac{(j-2)(j-3)}{2} \lambda_j$.*

The above estimate of the number of edges in the map graph of \mathcal{R} is too pessimistic. The next key lemma gives a better estimate.

Lemma 13. *Let F_1 and F_2 be two strongly touching faces of \mathcal{R} . Suppose that there are $r_{1,2}$ multiple edges between F_1 and F_2 in \mathcal{R}^* . Then, the following hold:*

1. *There is a set $X_{1,2}$ of at least $r_{1,2}$ vertices shared by the boundaries of F_1 and F_2 such that no two vertices of $X_{1,2}$ are shared by the boundaries of two faces F_3 and F_4 of \mathcal{R} with $\{F_1, F_2\} \neq \{F_3, F_4\}$.*

2. If we modify \mathcal{R} by triangulating F_1 and F_2 only, then the number of edges in the map graph of \mathcal{R} decreases by at most $\sum_{i=1}^2 \frac{(|V(F_i)|-2)(|V(F_i)|-3)}{2} - \frac{r_{1,2}^2 - 7r_{1,2} + 12}{2}$.

3. The map graph of \mathcal{R} has at most $3n_{\mathcal{R}} - 6 + \sum_{j=3}^k \frac{(j-2)(j-3)}{2} \lambda_j - \frac{k-280}{80} m_{\mathcal{R}}$ edges.

Proof. We prove the three statements in turn as follows.

(Statement 1) Consider the subgraph of \mathcal{R} whose edges are the edges shared by the boundaries of F_1 and F_2 and whose vertices are the endpoints of these edges. The subgraph is either a cycle of length $r_{1,2}$ or a collection of vertex-disjoint paths each of which has length at least 1. In the former case, we set $X_{1,2}$ to be the set of vertices on the cycle. In the latter case, we traverse the vertex-disjoint paths clockwise; during the traversal of each path, we let $X_{1,2}$ include all vertices except the last one on the path. Obviously, in both cases, $X_{1,2}$ is as required.

(Statement 2) By Lemma 12, triangulating F_1 (respectively, F_2) only causes the map graph of \mathcal{R} to lose at most $\frac{(|V(F_1)|-2)(|V(F_1)|-3)}{2}$ (respectively, $\frac{(|V(F_2)|-2)(|V(F_2)|-3)}{2}$) edges. For each $i \in \{1, 2\}$, call the quantity $\frac{(|V(F_i)|-2)(|V(F_i)|-3)}{2}$ the *pessimistic loss* of F_i .

Let \mathcal{K} be a k -face sphere graph obtained from \mathcal{R} by triangulating F_1 and F_2 only. Let E_1 (respectively, E_2) be the set of edges in \mathcal{K} between vertices in $V(F_1)$ (respectively, $V(F_2)$). Let u and v be two vertices in $X_{1,2}$. By the proof of Lemma 12, we have the following observations:

- Suppose that $E_1 \cup E_2$ contains an edge between u and v . Then, for each $i \in \{1, 2\}$, the edge $\{u, v\}$ is not counted in the pessimistic loss of F_i .
- Suppose that $E_1 \cup E_2$ contains no edge between u and v . Then, for each $i \in \{1, 2\}$, the edge $\{u, v\}$ is counted in the pessimistic loss of F_i .

Also note that there are at most $3|X_{1,2}| - 6$ (unordered) pairs $\{u, v\} \subseteq X_{1,2}$ such that $E_1 \cup E_2$ contains an edge between u and v . Thus, by the above observations, at least $\frac{|X_{1,2}|(|X_{1,2}|-1)}{2} - (3|X_{1,2}| - 6)$ edges are counted in both the pessimistic loss of F_1 and the pessimistic loss of F_2 . Hence, triangulating F_1 and F_2 only causes the map graph of \mathcal{R} to lose at most $\sum_{i=1}^2 \frac{(|V(F_i)|-2)(|V(F_i)|-3)}{2} - \frac{r_{1,2}^2 - 7r_{1,2} + 12}{2}$ edges.

(Statement 3) Let \mathcal{P} be the set of all (unordered) pairs $\{F_i, F_j\}$ such that F_i and F_j are strongly touching faces of \mathcal{R} . For each pair $\{F_i, F_j\} \in \mathcal{P}$, let $r_{i,j}$ be the number of edges shared by the boundaries of F_i and F_j in \mathcal{R} , and let $X_{i,j}$ be a set of at least $r_{i,j}$ vertices shared by the boundaries of F_i and F_j such that no two vertices of $X_{i,j}$ are shared by the boundaries of two faces $F_{i'}$ and $F_{j'}$ of \mathcal{R} with $\{F_i, F_j\} \neq \{F_{i'}, F_{j'}\}$. $X_{i,j}$ exists because of Statement 1.

By Statement 2, the pessimistic estimate (namely, $\frac{(|V(F_i)|-2)(|V(F_i)|-3)}{2} + \frac{(|V(F_j)|-2)(|V(F_j)|-3)}{2}$) of loss in the number of edges in the map graph of \mathcal{R} after triangulating F_i and F_j only, overcounts at least $\frac{r_{i,j}^2 - 7r_{i,j} + 12}{2}$ edges. We associate these overcounted edges with $X_{i,j}$.

Let $\{F_i, F_j\}$ and $\{F_{i'}, F_{j'}\}$ be two distinct pairs in \mathcal{P} . By our choices of $X_{i,j}$ and $X_{i',j'}$, the set of the overcounted edges associated with $X_{i,j}$ does not intersect the set of the overcounted edges associated with $X_{i',j'}$. Thus, by Lemma 12, the number of edges in the map graph of \mathcal{R} is at most $3n_{\mathcal{R}} - 6 + \sum_{j=3}^k \frac{(j-2)(j-3)}{2} \lambda_j - \sum_{\{F_i, F_j\} \in \mathcal{P}} \frac{r_{i,j}^2 - 7r_{i,j} + 12}{2}$. Note that $|\mathcal{P}| = m_{S^*}$ and $\sum_{\{F_i, F_j\} \in \mathcal{P}} r_{i,j} = m_{\mathcal{R}}$. Thus, by simple calculus, the number of edges in the map graph of \mathcal{R} is at most $3n_{\mathcal{R}} - 6 + \sum_{j=3}^k \frac{(j-2)(j-3)}{2} \lambda_j - \frac{m_{\mathcal{R}}^2}{2m_{S^*}} + \frac{7}{2}m_{\mathcal{R}} - 6m_{S^*}$, and hence is at most $3n_{\mathcal{R}} - 6 + \sum_{j=3}^k \frac{(j-2)(j-3)}{2} \lambda_j - \frac{k-280}{80}m_{\mathcal{R}}$ by Lemma 11. \square

Corollary 3. *The map graph of \mathcal{R} has at most $(\frac{79}{80}k + \frac{7}{2})n_{\mathcal{R}} - 2k$ edges.*

Lemma 14. *Suppose that $\sum_{j=3}^{k_2} (j-2)\lambda_{j,c} < \frac{n}{\beta}$. Then, the number of edges in the map graph of \mathcal{H} is less than $(\frac{79}{80} + \frac{11}{80\beta} + \frac{7(\beta-1)}{2\beta k})kn - 2k$.*

Proof. When constructing \mathcal{R} from \mathcal{H} , we deleted at most $\lfloor \frac{|V(F)|}{2} \rfloor$ vertices from each critical face F of \mathcal{H} . Thus, $n - n_{\mathcal{R}} \leq \sum_F \lfloor \frac{|V(F)|}{2} \rfloor$, where the summation is taken over all critical faces F of \mathcal{H} . In turn, since $|V(F)| \geq 3$, $n - n_{\mathcal{R}} \leq \sum_F (|V(F)| - 2) = \sum_{j=3}^{k_2} (j-2)\lambda_{j,c}$. So, $n_{\mathcal{R}} > \frac{\beta-1}{\beta}n$.

Now, by Lemma 9 and Corollary 3, the number of edges in the map graph of \mathcal{H} is at most $\frac{9}{8}k(n - n_{\mathcal{R}}) + (\frac{79}{80}k + \frac{7}{2})n_{\mathcal{R}} - 2k = \frac{9}{8}kn - (\frac{11}{80}k - \frac{7}{2})n_{\mathcal{R}} - 2k < (\frac{79}{80} + \frac{11}{80\beta} + \frac{7(\beta-1)}{2\beta k})kn - 2k$, because $k \geq 373$. \square

Setting $\beta = 41$ and combining Lemmas 7 and 14, we finally have Theorem 1. By Theorem 1 and Lemma 1, we have the following corollary:

Corollary 4. *Let $0 < \epsilon < \frac{3}{328}$, and let k be an integer not smaller than $\frac{140}{41\epsilon}$. Then, every k -map graph G with at least k vertices has at most $(\frac{325}{328} + \epsilon)kn - 2k$ edges, where n is the number of vertices in G .*

4 Lower Bounds

Theorem 2. *For every multiple n of 4 with $n \geq 8$, there is a 4-map graph with n vertices and $4n - 8$ edges.*

Theorem 3. *For every multiple k of 6, there are infinitely many integers n such that some k -map graph with n vertices has at least $(\frac{11}{12}k + \frac{1}{3})n$ edges.*

References

1. Z.-Z. Chen. Approximation algorithms for independent sets in map graphs. *J. Algorithms*, 41:20–40, 2001.
2. Z.-Z. Chen, M. Grigni, and C.H. Papadimitriou. Planar map graphs. *Proceedings of the 30th ACM Symposium on Theory of Computing*, pp. 514–523, 1998.
3. Z.-Z. Chen, M. Grigni, and C.H. Papadimitriou. Map graphs. *J. ACM*, 49:127–138, 2002.
4. D. P. Sanders and Y. Zhao. A new bound on the cyclic chromatic number. *J. Combinatorial Theory Ser. B*, 83:102–111, 2001.

Dynamic Storage Allocation and On-Line Colouring Interval Graphs

N.S. Narayanaswamy*

Department of Computer Science and Engineering
IIT-Madras, Chennai-600 036, India

Abstract. We present an improved on-line algorithm for colouring interval graphs with bandwidth. This problem has recently been studied in [1] and a 195-competitive online strategy has been presented. We improve this by presenting a 10-competitive strategy. To achieve this result, we use the online colouring algorithm presented in [8, 9]. We also present a new analysis of a polynomial time 3-approximation algorithm for Dynamic Storage Allocation(DSA) using features of the optimal on-line algorithm for colouring interval graphs [8, 9].

1 Introduction

Interval graphs are of interest in many fields as they are used to model the structure of many problems. One classical well studied problem is the issue of vertex colouring of interval graphs. Features of containment associated with intervals are well exploited to design efficient algorithms and fast implementations to optimally colour interval graphs (see [7]). Interval graphs are used to model many resource allocation problems. Each interval corresponds to a request for the usage of a resource exclusively for a certain period of time. In this paper we consider the issue of online colouring a set of intervals based on some relaxed properties (two adjacent vertices may get the same colour without violating an additionally specified constraint). Online algorithms are motivated by environments where the inputs occur in a sequence and need to be serviced without any knowledge of the future inputs. Online interval colouring algorithms are one of many such algorithms that are of much interest (see book [2]). Our results use an optimal on-line algorithm for colouring interval graphs presented in [8, 9].

The problem of colouring interval graphs with bandwidths is a generalization of the interval graph colouring problem. In the generalization, each interval has a weight in $(0, 1]$. These weights are referred to as bandwidths. A valid colouring of the vertices is one in which, for every r on the real line, the total bandwidth of the monochromatic intervals containing r is at most 1. Clearly, when each interval has bandwidth 1, we get the interval graph colouring problem. This problem has been of recent interest in [1] and, as has been remarked there, is a simultaneous generalization of online interval colouring and online bin- packing.

* Partly Supported by DFG Grant No. Jo 291/2-1. Part of Work done when the author was at the Institut Für Informatik, Oettingenstrasse 67, 80538, Munich, Germany.

We refer to a colouring satisfying the above condition as a *bandwidth satisfying colouring*. By a c -bandwidth satisfying colouring, we refer to a colouring which satisfies the property that the weight of a maximum clique is upper bounded by c in a graph induced by vertices assigned the same colour. A colouring that assigns different colours to the end points of each edge is simply referred to as a *valid colouring*.

An instance of DSA is $I = \{(d_1, s_1, r_1), \dots, (d_n, s_n, r_n)\}$ where the i -th interval, denoted by I_i , is $[s_i, r_i]$. d_i is a non-negative integer that can be visualized as the number of memory locations requested by a process which is alive during the time interval I_i . We use the word *demand* to refer to the memory requests. A solution to this problem instance is a vector of non-negative integers (b_1, \dots, b_n) such that, for $1 \leq i, j \leq n$, the intervals $[b_i, b_i + d_i]$ and $[b_j, b_j + d_j]$ are non intersecting whenever the intervals I_i and I_j have non-empty intersection. We call such an assignment (b_1, \dots, b_n) valid assignment and $\max_i (b_i + d_i)$ is called the *height* of (b_1, \dots, b_n) . The optimization problem is to find a minimum height valid assignment. The letters b, d, s , and r are chosen as they are suggestive of Base memory address, Demand, Starting time, and Release time, respectively. An instance of DSA is said to be *aligned* if all demands are powers of 2. An *aligned assignment* for an aligned instance of DSA is a valid assignment (b_1, \dots, b_n) in which each b_i is an integer and d_i divides b_i . Finally, DSA is a well motivated classical problem in computer science [10] studied in the settings of compile time memory optimization by Gergov [6], and bandwidth allocation by Leonardi et al. [11].

The representation of the input instance used in the DSA can be used to represent an instance for the bandwidth satisfying colouring problem by modifying d_i to be a number between $(0, 1]$. Geometrically, (d_i, s_i, r_i) represents an axis parallel rectangle of height d_i and width $r_i - s_i$. The input I can be visualized as an arrangement of rectangles, axis parallel unless mentioned otherwise, on the x -axis (referred to as the *time axis*). The required solution for the colouring problem is to assign to each interval a colour such that in the graph induced by vertices assigned the same colour, the maximum clique weight is at most 1. Geometrically, we can view each colour as corresponding to a unit height strip parallel to the time axis on the plane. Indeed, the strips corresponding to different colours are to be disjoint. The target is to use as few colours as possible. For DSA, the required solution is an arrangement in the plane such that rectangles that intersect along the time axis are disjoint and the position of rectangle (d_i, s_i, r_i) is obtained by translation along the y -axis (referred to as the *space axis*). The goal is to find a minimum height rectangle that contains this arrangement. In either case, we are required to translate the rectangles along the space axis to satisfy a related condition.

Previous Work: Online colouring algorithms for interval graphs have been well studied. The best known algorithm is in [8, 9] which uses at most $3\omega - 2$ colours. It is also shown to be an optimal online algorithm in [9]. Other approaches toward online colouring is to use the first fit algorithm: allot the smallest valid colour to each vertex. Online colourings with the bandwidth constraint have been studied recently [1]. Their work uses a combination of the optimal online

algorithm and the first fit approach. This algorithm achieves a constant, at most 195, competitive ratio.

For DSA, it is well known that finding the value of the optimum solution is NP-hard [4]. When the demands are all the same, the problem is the same as interval graph colouring which is well known to be polynomial time solvable [7]. On the other hand, the NP-hardness holds when each demand is an element of a set of two distinct integers. The best known approximation algorithms are a factor $\frac{5}{2}$ algorithm for aligned DSA and a factor 3 algorithm for DSA, both due to Gergov [5, 6].

Our Work and Results: We design a new online algorithm to colour interval graphs with the bandwidth constraint. Our approach is similar to that in [1] where the requests are classified into two subproblems based on the bandwidth. One subproblem is coloured using the first fit, and the other using the optimal on-line colouring algorithm. Each subproblem uses a distinct set of colours. In our approach, we classify the input into 3 classes, again based on bandwidth. We then appropriately apply the optimal on-line algorithm to each class, operating a different set of colours per class. The on-line algorithm is modified to suit the needs of each class, and this modification is what leads to an improved performance. The algorithm which is shown to be at most 10-competitive, a significant improvement.

Our result on the DSA stems from attempts to analyze the algorithm presented in Gergov's short paper [6]. Prior to this 3 approximation algorithm for DSA, the best known results were 6 approximation algorithms designed by Kierstead [8] and Slusarek [12]. The approach taken by Kierstead was to first convert an instance of DSA to an instance of aligned DSA by rounding each demand to the nearest larger or equal power of 2. Then the instance of aligned DSA was approximated within a factor of 3 using features of an on-line colouring algorithm. The two steps put together yield the claimed 6 approximation algorithm. In this work we use Kierstead's approach in an appropriate way to yield a 3-approximation algorithm for DSA. This algorithm, except for some minor differences, is same as the algorithm in [6]. Consequently, our contribution here is to present Gergov's work in the setting of the previous efforts at DSA, and an elegant proof along the same lines as Kierstead's analysis [8].

Plan: In Section 2 we present the required preliminaries and outline Kierstead's on-line colouring algorithm. The online algorithm for colouring with bandwidth constraint is present in Section 3. Finally, the approximation algorithm for DSA and its analysis is presented in Section 4.

2 Preliminaries

Without loss of generality, an interval is a set of consecutive non-negative integers. For a set of intervals $P = \{I_i : 1 \leq i \leq n\}$, we can associate a graph denoted by $G(P)$. In this graph, there are as many vertices as the number of intervals in P . Each interval is associated with a unique vertex and vice versa. Two vertices are adjacent if there is a non-empty intersection between the cor-

responding intervals. Similarly, for a set of rectangles we associate a rectangle graph.

The collection of intervals we are interested in are those that occur in an instance of the DSA. n denotes the number of intervals in an instance of DSA. For an instance $I = \{(d_i, s_i, r_i) : 1 \leq i \leq n\}$, $G(I)$ denotes the vertex weighted interval graph as defined previously. Clearly, a vertex weighted interval graph with non-negative integer weights yields an instance of the DSA. Also, the weighted intervals in I and weighted vertices in interval graphs correspond to rectangles, as mentioned in the introduction. Consequently, we use these notions interchangeably for ease of presentation.

For an undirected graph G , $\omega(G)$ denotes the size of the maximum cardinality clique in G . $\Delta(G)$ denotes the $\max\{\text{degree of } u : u \in V(G)\}$. For a weighted graph G , $\omega^*(G)$ denotes the largest weighted clique in G . $\chi^*(G)$ denotes the height of a valid assignment of minimum height. It is mentioned in Kierstead [8] that there is a weighted interval graph G for which $\chi^*(G) \geq \frac{5}{4}\omega^*(G)$. Note that in an unweighted interval graph (all weights equal to 1), $\chi(G) = \omega(G)$.

Let P be a collection of intervals. We now define three notions of *density* with respect to the collection P . For a positive integer r , the *density* of r is defined to be $D(r) = |\{I \in P : r \in I\}|$. The density of an interval I is defined as $D(I) = \min\{D(r) : r \in I\}$. The density of P , $D(P)$, is defined as $\max\{D(I) : I \in P\}$. We present the following crucial lemma from [8] for the sake of clarity.

Lemma 1. *Let P be a collection of intervals and $G = G(P)$. If $D(P) = 1$ then $\omega(G) \leq 2$ and $\Delta(G) \leq 2$.*

Proof. Since $D(P) = 1$, every interval I has a point r such that the point is present exclusively in I . That is, $D(I) = 1$ for every $I \in P$. Therefore, if $\omega(G) \geq 3$ then there exist 3 interval I_1, I_2, I_3 corresponding to vertices in G which form a triangle (a 3-clique). It follows that either $I_1 \subseteq I_2 \cup I_3$ or $I_2 \cup I_3 \subseteq I_1$. In either case we have identified an interval in P each of whose points is contained in another interval of P . This means there is an interval $I \in P$ such that $D(I) \geq 2$. This contradicts the hypothesis. Therefore, $\omega(G) \leq 2$. Similarly, a vertex of degree at least 3 implies that there is an interval which is contained in the union of at most two other intervals. This would violate the hypothesis that $D(P) = 1$. Therefore, $\Delta(G) \leq 2$.

2.1 On-Line Colouring Interval Graphs

It is well known that an interval graph can be coloured optimally with as many colours as the size of its largest clique. The algorithm first orders the vertices according to the leftmost point of the intervals associated with them. That is, the first vertex in the order is one whose leftmost point is the smallest. Then the vertices are considered according to the constructed order and greedily coloured. The colouring problem for interval graph becomes more challenging when we consider the problem of designing on-line algorithms. Here, along with the input interval graph, an order $\sigma = v_1, \dots, v_n$ is presented. The algorithm must colour the vertices according to σ and use as few colours as possible. Below we present

the on-line colouring algorithm due to Kierstead for interval graphs which uses at most $3\omega - 2$ colours.

Let $\sigma = v_1, \dots, v_n$ be the ordering of the vertices of G . The vertices of G are rectangles of unit height and length same as that of the interval associated with them. The algorithm can be visualized as running in two phases: In the first phase an arrangement of the rectangles satisfying certain properties is computed. In the second a rectangle graph of the arrangement is 3-coloured to obtain a colouring of the interval graph.

Phase I – Rectangle arrangement: Let v_i be the current vertex. Here we identify a position, $p(v_i)$, for v_i depending on $p(v_1), \dots, p(v_{i-1})$. First, for a non-negative integer r , let $G_r(v_i)$ be the induced subgraph of G on $\{v_j \in V(G) : j < i, p(v_j) \leq r, \{v_i, v_j\} \in E(G)\}$. $p(v_i)$ is the smallest non-negative integer r such that $\omega(G_r(v_i)) \leq r$. It is clear that for $r' < p(v_i)$, $\omega(G_{r'}(v_i)) > r'$. Pictorially, the rectangle corresponding to v_i sits on the line $p(v_i)$.

Properties of the Arrangement:

1. For each rectangle v , $p(v) \leq \omega - 1$.
2. Rectangles sitting on different lines do not intersect. This is true because the rectangles all have the same height.
3. For a number i , consider the collection P of intervals corresponding to the vertices of the induced graph on $\{v | p(v) = i\}$. This collection has density equal to 1. From lemma 1 it follows that maximum vertex degree in this graph is at most 2.

Phase-II: Colouring the intersection graph: A greedy colouring according to σ yields a 3-colouring. The reason is that the degree of each vertex in the rectangle graph is at most 2.

$p(v_i)$ depends only on the vertices which were considered prior to v_i and the colour assigned to v_i depends on the colour of its neighbours on the line $p(v_i)$. Further, vertices with $p(v) = 0$, all get the same colour, as they form an independent set. Consequently, it follows that the two phases can be performed on-line and at most $3\omega - 2$ colours are used.

3 Online Colouring Intervals with Bandwidth

Recall that the weight of each interval is a number in $(0, 1]$. The goal is to use the minimum number of colours to colour the vertices of an interval graph such that, for each colour c , the weight of the maximum weight clique is bounded by 1 in the graph induced by the vertices assigned c . The algorithm should be an on-line strategy, in the sense that at each time step (decided by the algorithm), an input pair consisting of an interval and its bandwidth requirement is presented to the algorithm. This request should be serviced by assigning a colour before the next request can be presented.

3.1 On-Line Strategy

The colours to be assigned are split into three classes C_1 , C_2 , and C_3 . Let $\sigma = v_1 v_2 \dots$ be the sequence of intervals. Let $b(v_i)$ denote the associated

bandwidth, $0 < b(v_i) \leq 1$. When an interval arrives it is processed, according to its bandwidth, as follows:

If $b(v_i) > \frac{1}{2}$: v_i is assigned a colour from the class C_1 . Any valid bandwidth satisfying colouring of the vertices is a valid colouring of the set of intervals. The reason being that each bandwidth is more than $\frac{1}{2}$. Applying Kierstead's on-line algorithm to colour v_i guarantees that the number of colours used is at most 3 times the optimum number of colours.

If $\frac{1}{4} < b(v_i) \leq \frac{1}{2}$: In this case v_i is coloured using colours from C_2 . The colouring strategy described below uses as many colours as the size of the largest clique formed by intervals whose bandwidth is in $(\frac{1}{4}, \frac{1}{2}]$.

Colouring Strategy: Ignore the bandwidth requirements, that is treat all these bandwidths as 1, and apply only the rectangle arrangement phase of the online colouring algorithm. ω colours are used from C_2 to colour the vertices based on this arrangement: for a vertex v with $p(v) = y$, the y -th colour is assigned. As the bandwidths are at most $\frac{1}{2}$, we have obtained a bandwidth satisfying colouring. Recall that ω denotes the size of the largest clique in the graph induced by the intervals in question.

Lemma 2. *The above strategy outputs a bandwidth satisfying colouring using at most ω colours.*

Proof. As argued in section 2.1, for each $y \in \{0, 1, \dots, \omega - 1\}$, the interval graph formed by intervals in the set $\{v | p(v) = y\}$ does not have a clique with more than 2 vertices. Further, for each v , $p(v) \leq \omega - 1$. Clearly, at most ω colours are used, and the colouring is bandwidth satisfying as each bandwidth requirement is at most $\frac{1}{2}$. Hence the lemma is proved.

To compare with the optimum bandwidth satisfying assignment, we lower bound the number of colours in an optimum bandwidth satisfying colouring in terms of the number of colours in an optimum valid colouring.

Lemma 3. *The number of colours used by an optimum bandwidth satisfying colouring is at least one-third the number of colours used in an optimum valid colouring which is at least ω .*

Proof. In any optimum bandwidth satisfying solution, the largest clique size among monochromatic vertices is 3. If it exceeds 3, then such a solution wouldn't be feasible, as the clique weight would be more than 1 due to the bandwidths being more than $\frac{1}{4}$. To obtain a valid colouring, we optimally colour each colour class using at most 3 colours. The number of colours used is at most 3 times the optimum bandwidth satisfying colouring. Therefore, the optimum valid colouring is at most 3 times the optimum bandwidth satisfying colouring. In other words, an optimum bandwidth satisfying colouring uses at least one-third the number of colours used by an optimum valid colouring.

Number of Colours Used: From the previous two lemmas it follows that the on-line algorithm uses at most thrice the number of colours used by an optimum bandwidth satisfying colouring. The above result is a specific case of a more general algorithm called STEADY in [3].

If $b(v_i) \leq \frac{1}{4}$: These requests are assigned colours from the set C_3 . Here we modify the on-line colouring algorithm due to Kierstead presented in section 2.1.

Phase I – Rectangle arrangement: Let v_1, \dots, v_{i-1} denote the vertices corresponding to the intervals preceding v_i . Here we identify a position $p(v_i)$, for v_i , depending on $p(v_1), \dots, p(v_{i-1})$. First, for a non-negative real number r , let $G_r(v_i)$ be the induced subgraph of G on $\{v_j \in V(G) : j < i, p(v_j) \leq r, \{v_i, v_j\} \in E(G)\}$. $p(v_i)$ is the smallest real number r such that $\omega^*(G_r(v_i)) \leq r$. It is clear that for $r' < p(v_i)$, $\omega^*(G_{r'}(v_i)) > r'$. Again, pictorially, the rectangle corresponding to v_i sits on the line $p(v_i)$.

Lemma 4. *For a number y , consider the collection P_y of intervals corresponding to the vertices of $\{v | y \in [p(v), p(v) + b(v)]\}$. For all y , $D(P_y) = 1$. In other words, the collection P_y has density equal to 1.*

Proof. Let us assume that the statement is false. Consider the smallest y such that $D(P_y) > 1$. Let $v_i \in P_y$ be such that for each $x \in I(v_i)$, $D(x) \geq 2$. Let r denote $p(v_i)$. Clearly, $\omega^*(G_r(v_i)) \leq r$, and for all $r' < r$, $\omega^*(G_{r'}(v_i)) > r'$. Consequently, there is a clique K in $G_r(v_i)$ whose weight is r . In particular, there is a point $z \in I(v_i)$ which is common to the intervals corresponding to the vertices in K . Further, $D(z) = 1$; only $I(v_i) \in P_r$ contains z , otherwise weight of K would be greater than r , and $\omega^*(G_r(v_i)) > r$, which is a contradiction to the fact that $p(v_i) = r$. On the other hand, $D(z) \geq 2$ in P_y . Let y' be the smallest number in $[p(v_i), y]$ for which $D(z) = 2$ in $P_{y'}$. Clearly, there is a $v_j, j > i$ such that $p(v_j) = y'$ and $z \in I(v_j)$. Therefore, $K \cup v_i \in G_{y'}(v_j)$. Consequently, $\omega^*(G_{y'}(v_j)) \geq r + b(v_i) = p(v_i) + b(v_i) > y \geq y'$. This contradicts the rule for the choice of $p(v_j)$, and this has arisen because of our assumption that there is a y for which $D(P_y) > 1$. The assumption is false, and hence the lemma is proved.

Lemma 5. *The number of colours required by a bandwidth satisfying assignment is lower bounded by the ceiling of the height, H , of the arrangement output by phase-I.*

Proof. It is clear that there is a clique K of weight H . To see this consider a v_i such that $p(v_i) + b(v_i) = H$. Let $r = p(v_i)$. By the argument used in lemma 4, there is a clique K in $G_r(v_i)$ whose weight is $r = p(v_i)$. $K \cup v_i$ is also a clique, and has weight $r + b(v_i) = p(v_i) + b(v_i) = H$. In a bandwidth satisfying assignment, the set of vertices of K that get the same colour form a clique of weight at most 1. Consequently, the number of distinct colours used to colour the vertices of K must be at least as many as the weight of the clique, which is the height of the arrangement. Therefore, the number of distinct colours used is at least the ceiling of H . Hence the lemma.

Phase-II: Colouring the intersection graph: The colouring approach is as follows: if $\frac{1}{4} \leq p(v_i) < \frac{1+1}{4}$, then v_i is allocated the colour $c_j \in C_3$.

Analysis: Clearly, the above scheme is an on-line strategy, as it processes a request fully before considering the next. Our interest is in the number of colours that are used by the strategy, and the correctness of the result. The number of colours used is at most 4 times the optimum. The reason is that we split the

rectangle arrangement output by phase-I into blocks of height $\frac{1}{4}$ each. The number of such blocks is 4 times the height of the arrangement, and each block gets a separate colour. Consequently, the number of colours used is upper bounded by 4 times the ceiling of the height. The ceiling of the height of the arrangement is a lower bound on the number of colours used by any bandwidth satisfying colouring. Therefore, we use at most 4 times the optimum number of colours.

Correctness. We now show that in the graph induced by the intervals that have been assigned the same colour, the weight of any clique is at most 1. Let us assume the contrary, that there is a clique of height greater than 1 among vertices assigned the same colour, say c_j . Since this clique is formed by intersecting intervals, there is one point common to all these intervals. Let this point be t . In the arrangement output by the above procedure, there is a point (t, u) , $\frac{j}{4} \leq u < \frac{j+1}{4}$ which is occupied by at least 3 rectangles. Otherwise, using the fact that all the bandwidths are at most $\frac{1}{4}$, it follows that sum of bandwidths at point t is at most 1. Let us consider the collection P_u . Recall that P_u is the set of intervals corresponding to the set $\{v | u \in [p(v), p(v) + b(v))\}$. Let I_1, I_2 , and I_3 be three intervals in P_u that contain the point t . A simple property of a set of intervals is that if three intervals share a point, then one of the intervals is contained in the union of the other two. Let us assume without loss of generality, that $I_1 \subseteq I_2 \cup I_3$. Clearly, $D(I_1) \geq 2$, and $D(P_u) > 1$. This conclusion contradicts lemma 4. Our assumption is wrong and therefore, it follows that the weight of any clique formed by vertices assigned the same colour c_j is at most 1.

Completing the Analysis: We have split the input sequence into three cases. We have also shown that Kierstead's on-line colouring algorithm can be modified to perform competitively on each case. If the optimum for the 3 cases were O_1, O_2 , and O_3 respectively, then the optimum number of colours used by a bandwidth satisfying colouring is lower bounded by each of the 3 optima. Using this fact, it follows that the on-line algorithm outlined above uses at most 10 times the optimum number of colours used by a bandwidth satisfying assignment.

4 Dynamic Storage Allocation

Consider an instance $I = \{v_i = (d_i, s_i, r_i) | 1 \leq i \leq n\}$ of DSA. The interval graph colouring problem is a special case of the DSA in which all the demands are the same. Our idea is to consider the vertices in a certain order such that the rectangle graph obtained from an appropriate modification, to take into account non-trivial vertex weights, of Phase-I of the above online colouring algorithm is 3-colourable. In the case when all demands are same, this property is guaranteed for any order in which we consider the vertices. The algorithm is as follows:

Initially $p(v_i)$ is undefined for all $v_i \in I$ and *base* is a variable initialized to 0.

1. While there are unassigned rectangles, do the following steps 2 to 4. Let A be current set of vertices for which $p(v_i)$ is not undefined.
2. For a non-negative integer r , let $G_r(v_i)$ be the vertex weighted graph induced by $\{u | u \in A, \{v_i, u\} \in E(G), p(u) \leq r\}$. A vertex $u \in G_r(v_i)$ has weight $w_u = r + 1 - p(u)$ if $r + 1 - p(u)$ is at most the demand made by u . Otherwise, w_u is the demand made by u . $a(v_i)$ is the smallest $r \geq \text{base}$ such that $\omega^*(G_r(v_i)) \leq r$.

3. Consider a vertex v such $a(v)$ is the least among all vertices not in A .
4. Add v to A and $p(v)$ is set to $a(v)$. $base := p(v)$.
5. If there are some rectangles outside A , go back to step 2.

We now prove a lemma that is used to conclude that the rectangle graph of the above arrangement is indeed 3-colourable. Below, we use A to denote the arrangement output by the above algorithm for an instance I . We use $R(v)$ to denote the rectangular region in A associated with v . For v_i , $R(v_i)$ is the rectangle whose diagonal is the line joining the points $(s_i, p(v_i))$ and $(r_i, p(v_i) + d_i)$. Geometrically, in A , the rectangle v_i sits on the line $y = p(v_i)$. v_i is said to be *assigned* to line $p(v_i)$.

Lemma 6. *For an integer r , the graph induced by $\{v | p(v) \leq r \text{ and } R(v) \cap \{(x, y) | x \in I(v), y = r + 1\} \neq \emptyset\}$ has maximum vertex degree upper bounded by 2.*

Proof. For our proof, we view a vertex of demand d as d rectangles of unit demand. For notation, the rectangles associated with v_i are $v_{i,1}, \dots, v_{i,d_i}$ and for each $j, 1 \leq j \leq d_i$, the interval associated with $v_{i,j}$ is $[s_i, r_i]$. Recall that $[s_i, r_i]$ is the interval associated with v_i . Let H denote this new interval graph. Also, it is clear that A yields a rectangle arrangement for H . In this arrangement, $p(v_{i,j}) = p(v_{i,1}) + j - 1 = p(v_i) + j - 1, 1 \leq i \leq n, 1 \leq j \leq d_i$. For an integer t , let $H_t(v_{i,j})$ be the induced subgraph of H on the vertex set $\{u \in H : p(u) \leq t, \{u, v_{i,j}\} \in E(H)\}$. Now, it is easy to observe the following for each $i, j, 1 \leq i \leq n, 1 \leq j \leq d_i$, by induction on j :

- If $t < p(v_{i,j})$, $\omega(H_t(v_{i,j})) > t$.
- If $t = p(v_{i,j})$, $\omega(H_t(v_{i,j})) \leq t$.

Now we show that for each t , the density of the set of intervals $\{u : u \in H, p(u) = t\}$ is equal to 1. From lemma 1 it follows that the maximum vertex degree in the corresponding interval graph is at most 2. This conclusion immediately implies this lemma. We now prove this claim about the density by contradiction. Let us assume that for some t , the density of the set $X = \{u : u \in H, p(u) = t\}$ is more than 1. Let $u_1 \in X$ be an interval whose density is more than 1. Since $p(u) = t$, by definition, $\omega(H_{t-1}(u_1)) \geq t$. Let K be a clique of this size from $H_{t-1}(u_1)$. Let r be a point which belongs to the interval u_1 and the intervals that form K . Since u_1 has density more than 1, r is in another interval $u_2 \in X$. Therefore, u_1, u_2 and K form a $t + 2$ clique. Consequently, if u_1 was assigned to A before u_2 , then $H_t(u_2)$ contains the clique $K \cup u_1$. This is a contradiction to the fact that $p(u_2) = t$, since $\omega(H_t(u_2)) > t$. We arrive at a similar contradiction if we consider the case where u_2 was assigned to A before u_1 , then we have a contradiction to the fact that $p(u_1) = t$. Therefore, our assumption is wrong. The density of X is 1. Hence the proof of our claim and consequently, the lemma.

Height of the Arrangement: This height is bounded by the size of a maximum weighted clique in the weighted interval graph associated with I . Let B be the height of the arrangement. Let v_i be a rectangle which has a non-empty intersection with the line $\{(x, y) | y = B\}$. By definition of $p(v_i)$, it follows that

there is a clique of weight $B - d_i$ in the neighbourhood of v_i . Consequently, there is a clique of weight B in $G(I)$. Therefore, B is at most the size of a maximum weighted clique.

Storage Allocation from A: The solution for I is obtained by greedily colouring the rectangle intersection graph obtained from A . The vertices are considered for colouring in the order in which they were entered into A . The fact that greedy colouring uses at most 3 colours follows from 6. This yields a solution whose height is at most 3 times the size of a maximum weighted clique. Hence we have a 3-approximation algorithm for DSA.

Acknowledgments

Thanks to Rohit Khandekar, Sunil Chandran, Thomas Erlebach, and Udo Adamy. Thanks to Jan Johannsen for his encouragement and support.

References

1. Udo Adamy and Thomas Erlebach. Online coloring of intervals with bandwidth. In *Proceedings of the First Workshop on Approximation and Online Algorithms*, 2003.
2. A. Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
3. A. Feldmann, B. Maggs, J. Sgall, D.D. Sleator, and A. Tomkins. Competitive analysis of call admission algorithms that allow delay. Technical Report CMU-CS-95-102, Carnegie Mellon University, 1995.
4. M.R. Garey and D.S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. Freeman, 1979.
5. J. Gergov. Approximation algorithms for dynamic storage allocation. In *Proc. of European Symposium on Algorithms*, volume 1136 of *LNCS*, pages 52–61. Springer, 1996.
6. J. Gergov. Algorithms for compile-time memory optimization. In *Proc. of 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 907–908, 1999.
7. M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, 1980.
8. H.A. Kierstead. A polynomial approximation algorithm for dynamic storage allocation. *Discrete Mathematics*, 88, pages 231–237, 1991.
9. H.A. Kierstead and W.T. Trotter. An extremal problem in recursive combinatorics. In *Congressus Numeratum*, volume 33, pages 143–153, 1981.
10. D.E. Knuth. *Art of Computer Programming, Vol.1: Fundamental Algorithms*. Addison-Wesley, 1973.
11. S. Leonardi, A. Marchetti-Spaccamela, and A. Vitaletti. Approximation algorithms for bandwidth and storage allocation problems under real time constraints. In *Proceedings of the 20th Foundations of Software Technology and Theoretical Computer Science*, volume 1974 of *LNCS*, pages 409–420. Springer, 2000.
12. M. Slusarek. A colouring algorithm for interval graphs. In *Proc. of 14th Mathematical Foundations of Computer Science*, volume 379 of *LNCS*, pages 471–480. Springer, 1989.

New Approximation Algorithms for Some Dynamic Storage Allocation Problems^{*}

Shuai Cheng Li¹, Hon Wai Leong¹, and Steven K. Quek²

¹ School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543
{lisc,leonghw}@comp.nus.edu.sg
² School of Information Technology
Nanyang Polytechnic, Singapore
steven_quek@nyp.edu.sg

Abstract. The offline dynamic storage allocation (DSA) problem has recently received some renewed attention and several new results have been reported. The problem is NP-complete and the best known result for the offline DSA is a polynomial time 3-approximation algorithm [Gerg99]. Better ratios have been reported for special cases if restrictions are placed on the allowable sizes of the blocks [Gerg96,MuBh99]. In this paper, we present new techniques for solving special cases with *blocks of restricted sizes* and we obtain better approximation ratios for them. We first obtain results for small instances which are then used to solve the more general cases. Our main results are (i) a $4/3$ -approximation algorithm when the maximum block size $h=2$ (previous best was $3/2$); and (ii) a 1.7 -approximation algorithm for the case $h=3$ (previous best was $1\frac{11}{12}$).

1 Introduction

The *dynamic storage allocation* (DSA) problem is a classic combinatorial optimization problem that has a wide variety of applications, including dynamic memory allocation in operating systems ([Robs74,Knut73,WJNB95]).

Our study an offline DSA is motivated by the *berth allocation problem* (BAP) in busy container transshipment ports (such as Singapore). Ships arriving at the port have fixed arrival and departure times and have to be berthed with a (straight-line) section of the port so that they can unload and load containers. Each vessel must be berth *entirely* within the section. Effective allocation of berthing space to the ships is an important planning task in many ports in order to increase throughput, reduce delay, and optimize the use of scarce resources (wharf areas, cranes).

We show that the BAP is precisely the offline DSA problem. This DSA problem was first studied by Knuth, Robson and others [Robs74,Knut73] and has

^{*} This work was supported in part by the National University of Singapore under Grant R252-000-128-112.

recently received some renewed attention. The problem is NP-complete [GJ79] and there were a series of approximation algorithms proposed in recent years [Kier88,Kier91,Gerg96,Gerg99]. The best known result is a 3-approximation algorithm by Gergov [Gerg99]. Better ratios have been reported for special cases if restrictions are placed on the allowable sizes of the blocks [Gerg96,MuBh99].

In this paper, we present new techniques for solving more special cases (of blocks with restricted sizes) that produce better approximation ratios. Our motivation for studying special case are twofold: Firstly, we hope that results of small special cases will lead to better approximation algorithms for these special cases. Secondly, our research [Li02] has shown that these approximation algorithms for special cases can be adapted to solve the general case and produce good results in practice for the berth allocation problem.

2 Problem Formulation and Previous Results

The offline DSA problem is formulated as follows¹: An instance \mathcal{B} of the *offline* DSA is given by a set of blocks $\mathcal{B} = \{B_1, B_2, \dots, B_n\}$. Each block B_j ($1 \leq j \leq n$) is a triple of nonnegative integers (s_j, a_j, d_j) where s_j is the *size* of the requested block while a_j and d_j are the *arrival time* and *departure time* of B_j . A (memory) allocation $b(\mathcal{B}) = (b_1, b_2, \dots, b_n)$ is a mapping that assigns to each block B_j a nonnegative integer *location* b_j such that for all i, j , ($1 \leq i < j \leq n$), either $[a_i, d_i] \cap [a_j, d_j] = \emptyset$ or $(b_i, b_i + s_i) \cap (b_j, b_j + s_j) = \emptyset$. Note that in the online version of the DSA, we do not know \mathcal{B} in advance.

The DSA can be geometrically interpreted in 2-D (as illustrated in Figure 1) where the x -axis represents the time axis and the y -axis represents the linear storage. The block B_j is represented by a rectangle whose x -extend is the time interval $[a_j, d_j)$ and the y -extend is the (contiguous) space allocation denoted by $(b_j, b_j + s_j)$. The DSA problem is then to pack the collection of rectangles given by $\mathcal{B} = \{B_1, B_2, \dots, B_n\}$ into a larger rectangle. Since the x -extend of each block is fixed, the rectangles are only free to move vertically and the allocation function b merely determined their y positions.

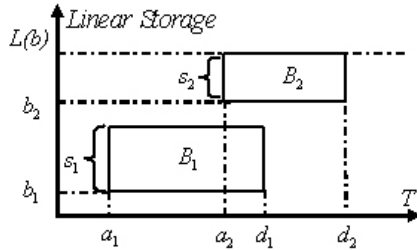


Fig. 1. A DSA instance with two blocks.

For any allocation $b(\mathcal{B})$, we define the *cost* $L(b) = \max_{(1 \leq j \leq n)} \{b_j + s_j\}$. Geometrically, we can interpret $L(b)$ to be the minimum height of a rectangle

¹ Our notations are similar to those of [Gerg96].

that contains all the allocated rectangles. We distinguish two versions of the DSA. In the *optimization* version of the DSA, we are given an instance \mathcal{B} of the DSA and we want to find a minimum cost allocation, namely, an allocation b that minimizes $L(b)$. In the *decision version* of the DSA, we are given an instance \mathcal{B} and a cost threshold L_0 and we want to find an allocation $b(\mathcal{B})$ with $L(b) \leq L_0$.

The berth allocation problem (BAP) is precisely the decision version of the offline DSA problem if we model each ship by a block $B_j = (s_j, a_j, d_j)$, where the size s_j corresponds to the length of the ship, while a_j and d_j correspond to the arrival time and departure time of the ship. The cost threshold L_0 corresponds to the length of the section. Then the allocation b_j corresponds to the allocation the ship to the location $(b_j, b_j + s_j)$ along the section. An allocation $b(\mathcal{B})$ with $L(b) \leq L_0$ corresponds to a BAP solution where all the ships are berthed without overlap within a section of length L_0 .

We note that the DSA is analogous to a 2-D rectangle packing problem – that of packing the smaller rectangles into the bigger rectangle. In Figure 2, we illustrate this viewpoint. We refer to the larger rectangle as the *packing area*. Allocating a rectangle can also be viewed as “packing” the rectangle into the packing area. For the rest of the paper, we shall use the term “allocate” and “pack” interchangeably.

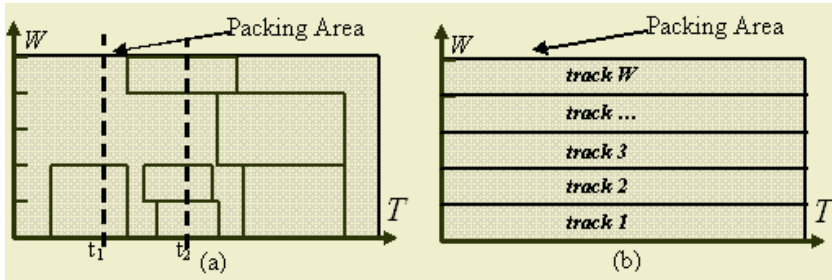


Fig. 2. The DSA as a 2D packing problem. (a) A solution with width $W = 5$, $m_{t_1}(\mathcal{B}) = 2$, $m_{t_2}(\mathcal{B}) = 3$, and $m(\mathcal{B}) = 4$. (b) Partitioning the packing area into 5 tracks.

A block B_j is said to be *active at time t* if $t \in [a_j, d_j]$, or equivalently, if $a_j \leq t < d_j$. Let $B(t)$ denote the set of blocks that are active at time t , namely, $B(t) = \{B_j : t \in [a_j, d_j]\}$. Then, the *local density at time t* (denoted by $m_t(\mathcal{B})$) is defined to be the total size of the active blocks at time t , namely, $m_t(\mathcal{B}) = \sum_{B_j \in B(t)} s_j$. The *density* of the instance \mathcal{B} (denoted by $M(\mathcal{B})$) is the maximum local density, namely, $M(\mathcal{B}) = \max_t \{m_t(\mathcal{B})\}$. Given an instance \mathcal{B} of the DSA, it is easy to see that $L(b) \geq M(\mathcal{B})$ for any allocation $b(\mathcal{B})$.

The DSA problem is NP complete and there has been a number of approximation algorithms proposed. This ratio has been steadily reduced from $\Theta(\ln h)$ [Robs74], and to 80 by Kierstead [Kier88]. Slusarek [Slus89], and independently, Kierstead [Kier91] reduced the ratio to 6. Subsequently, this ratio was reduced to 5 by Gergov [Gerg96] and later to 3 [Gerg99]. The latter result is the best-known general result to date.

Definition 1. Let $DSA(W, M, h)$ denote the class of DSA instances \mathcal{B} where
 (1) the maximum block size is h (namely, for all j , $1 \leq s_j \leq h$),
 (2) the density of the instance is M ($M(\mathcal{B}) = M$), and
 (3) the cost threshold $L_0 = W$.

Definition 2. An instance $\mathcal{B} \in DSA(W, M, h)$ is said to be feasible if there exists an allocation $b(\mathcal{B})$ with $L(b) \leq W$. The class $DSA(W, M, h)$ is said to be feasible if all instances in the class are feasible.

Then, the optimization version of the DSA problem can be formulated as follows: Let $\mathcal{L}(M, h)$ denote the smallest W such that $DSA(W, M, h)$ is feasible. For a fixed h , let $\mathcal{L}(h) = \lim_{M \rightarrow \infty} \frac{\mathcal{L}(M, h)}{M}$. We have the following results:

Theorem 1. [GJ79] $\mathcal{L}(M, 1) = M$ for all $M \geq 1$. Namely, $\mathcal{L}(1) = 1$

Theorem 2. [Gerg99] $\mathcal{L}(M, h) \leq 3M$ for all $h \geq 1$. Namely, $\mathcal{L}(h) \leq 3$

Results for several special cases have also been reported. For the DSA variant with two distinct block sizes (say s_1 and s_2), Gergov [Gerg96] showed that $\mathcal{L}(M, \{s_1, s_2\}) \leq 2M$ or $\mathcal{L}(\{s_1, s_2\}) \leq 2^2$. For the case of blocks with sizes 1 and h , it was shown [MuBh99] that the First Fit (FF) algorithm has approximation ratio $2 - 1/h$, namely, $\mathcal{L}(\{1, h\}) \leq (2 - 1/h)$. Note that this implies $\mathcal{L}(2) \leq 1\frac{1}{2}$.

For the online DSA problem, we let $ODSA(W, M, h)$ denote the class of instances in which all the blocks have sizes $\leq h$ and the local density at any time t is bounded by M and $L_0 = W$. Let $\mathcal{N}(M, h)$ denote the smallest W such that $ODSA(W, M, h)$ is feasible. For a fixed h , let $\mathcal{N}(h) = \lim_{M \rightarrow \infty} \frac{\mathcal{N}(M, h)}{M}$. For the general case, the best result, due to Robson [Rob74], is that $0.5M \log_2 h \leq \mathcal{N}(M, h) \leq 0.84M \log_2 h$. For small values of h , it has been shown that the following holds [Knut73, Rob74, LuNO74]:

Theorem 3. For the online DSA, we have

- (a) $\mathcal{N}(2) = 1.5$, and more precisely, $\mathcal{N}(M, 2) = \lceil (3M - 1)/2 \rceil$,
- (b) $1\frac{2}{3} \leq \mathcal{N}(3) \leq 1\frac{11}{12}$,
- (c) $\mathcal{N}(\{1, h\}) = 2 - 1/h$.

In this paper, we study $DSA(W, M, h)$ and obtain values of $\mathcal{L}(M, h)$ for small M and $h = 2$ and 3 using new algorithms for solving these basic cases. We then present new approximation algorithms based on channel partitioning that achieves improved bounds for $\mathcal{L}(h)$, $h = 2, 3$. Both of these bounds are superior to the corresponding best known value for $\mathcal{N}(h)$ (the online case). For $h = 2$, it is known that $\mathcal{L}(2) \geq \frac{5}{4}$ while the best known upper bound is $\frac{3}{2}$, which is the same as $\mathcal{N}(2)$. We improve this to $\mathcal{L}(2) \leq \frac{4}{3}$. For $h = 3$, we show that $\mathcal{L}(3) \leq 1.7$ while the best known bound for $\mathcal{N}(3)$ is $1\frac{11}{12}$.

² Here, we extend the notation $DSA(W, M, h)$ to $DSA(W, M, S)$, where the S denote the set of allowable block sizes. We also extend the notation of \mathcal{L} accordingly.

3 Solving Some Small Basic Cases

We first solve $DSA(W, M, h)$ classes with some small M and $h = 2, 3$. In solving these cases, we develop algorithms that partition the packing area into two adjacent channels and use a *channel flip procedure* for effective allocation. These ideas are central to the approximation algorithms developed in the next section for $\mathcal{L}(2)$ and $\mathcal{L}(3)$.

In our packing algorithm for small DSA instances, it is convenient to divide the packing area into horizontal *tracks* of unit length as illustrated in Figure 2(b). For $DSA(W, M, h)$, we have tracks $1, 2, \dots, W$, where track k refer to the horizontal strip whose y -extend is $(k-1, k)$. For a block B_j (with size s_j), setting $b_j = k$ is equivalent to allocating the (contiguous) tracks $k+1, k+2, \dots, k+s_j$ to B_j , or equivalently, the y -extend of B_j is $(k, k+s_j)$. For example, allocating a (unit size) block B_j to track 1 means $b_j = 0$.

3.1 Basic Cases for $h = 2$

Theorem 4. *For the offline DSA problem, we have*

$$(a) \mathcal{L}(2, 2) = 2, \quad (b) \mathcal{L}(3, 2) = 3, \quad (c) \mathcal{L}(4, 2) = 5, \quad (d) \mathcal{L}(5, 2) = 6.$$

It is easily shown that $DSA(2, 2, 2)$ is feasible using a First Fit algorithm. The algorithm for solving $DSA(3, 3, 2)$ is a modified version of online first fit which we call *First-Fit with Channel Flip* (FF-CF). We first divide the 3 tracks into two channels C_1 with 1 track and C_2 with two contiguous tracks. There are two possible configurations: (a) $C_1 = \{1\}$, $C_2 = \{2, 3\}$ or (b) $C_1 = \{3\}$, $C_2 = \{1, 2\}$. The actual configuration in use will vary during the allocation process as described below.

The algorithm FF-CF processes the blocks in \mathcal{B} in increasing order by their arrival times. The blocks are packed (allocated) using a modified first fit algorithm trying channel C_1 first, and then C_2 . With loss of generality, assume that the configuration used is $C_1 = \{1\}$, $C_2 = \{2, 3\}$. In a general step, we want to pack a block B that arrives at time t . We distinguish the following cases:

Case 1: If B has size 1, B is allocated to the first free track using first fit.

Case 2(a): If B has size 2 and channel C_2 is free, we allocate B to C_2 (tracks 2, 3).

If B has size 2 and channel C_2 is not free, then C_1 must be free and there is exactly one occupied track in C_2 (since local density at time $t \leq 3$). There are two sub-cases:

Case 2(b): If the middle track (track 2) of C_2 is free, then both tracks 1 and 2 are free. We perform a *channel-flip* operation at time t in which we change the channel configuration to $C_2 = \{1, 2\}$ and $C_1 = \{3\}$. We then allocate block B to the new C_2 .

Case 2(c): If the middle track (track 2) of C_2 is occupied, then the two free tracks (tracks 1 and 3) are “split” by the middle track. This is illustrated in Figure 3. In this case, we “exchange” the two tracks in channel C_2 – namely, exchange the allocation of all the size 1 blocks in C_2 that arrives after the most

recent size 2 block. After the exchange operation, tracks 1 and 2 are free at time t and this case reduces to Case 2(b) above. We perform a *channel-flip* operation at time t and allocate B to the new C_2 .

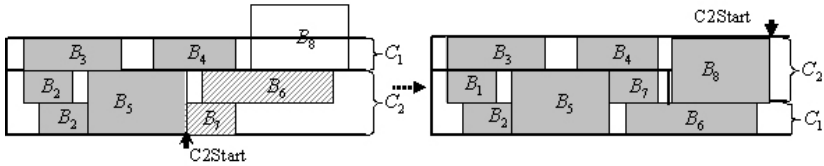


Fig. 3. Example of track exchange and Channel Flip operations for DSA(3,3,2). (a) Channel C_1 and C_2 before block B_8 is assigned; tracks 1 and 3 are free. (b) The configuration after exchanging the “active size 1 blocks” in C_2 (blocks B_6 and B_7), and performing a channel flip. The block B_8 is assigned to the new C_2 .

Algorithm 1: DSA-Alg(3,3,2, \mathcal{B}): [FF-CF Algorithm]

```

1. begin
2.   Sort the blocks in  $\mathcal{B}$  by arrival time;
3.   Define channels  $C_1 = \{1\}$  and  $C_2 = \{2, 3\}$ ;
4.    $FS \leftarrow \emptyset$ ,  $C2Start \leftarrow 0$ 
5.   for each block  $B_j \in \mathcal{B}$  do
6.     if ( $s_j = 1$ ) then
7.       Pack  $B_j$  at the first free track using First Fit;
8.       if ( $B_j$  is packed into  $C_2$ ) then  $FS \leftarrow FS \cup \{B_j\}$ ;
9.     else // ( $s_j = 2$ )
10.      if ( $C_2$  is free)
11.        then Pack  $B_j$  into  $C_2$ ;
12.      else Channel-Flip( $FS$ ,  $B_j$ );
13.       $C2Start \leftarrow d_j$ ;  $FS \leftarrow \emptyset$ ;
14.   end
15.   Procedure Channel-Flip( $FS$ ,  $B_j$ );
16.   begin
17.     if (middle track of  $C_2$  is occupied)
18.       then Exchange the track allocation of the blocks in  $FS$  on  $C_2$ ;
19.     Flip the channel configuration of  $C_1$  and  $C_2$ ;
20.     Pack  $B_j$  in the new  $C_2$ ;
21.   end {Flip}

```

The details of the algorithm is shown in Algorithm 1. Correctness of the algorithm is easily shown by proving the invariant that every size channel-flip operation generates a new empty C_2 channel that is used to pack a size 2 block. The running time for this algorithm is $O(n \log n)$, since we need to sort all the blocks, and each block would get allocated once, and flipped at most once.

Proof of Theorem 4(c): To prove Theorem 4(c), namely that $\mathcal{L}(4, 2) = 5$, we first prove that DSA(4,4,2) is infeasible. Figure 4 shows an instance of DSA(4, 4, 2) that can be shown to be not feasible (using a case analysis).

Lemma 1. $DSA(4, 4, 2)$ is not feasible.

Corollary 1. $DSA(M, M, h)$ is infeasible for all $h \geq 2$, and $M \geq 4$.

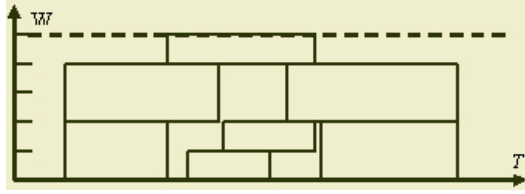


Fig. 4. An infeasible $DSA(4, 4, 2)$ instance. A careful case analysis shows that this instance requires 5 tracks, namely $W = 5$. It also gives a lower bound of $\frac{5}{4}$ for $\mathcal{L}(2)$.

Next, we show that $DSA(5, 4, 2)$ is feasible by giving a constructive algorithm called *First-Fit with Block Partitioning* (FF-BP). Instead of partitioning the tracks, the algorithm partitions the blocks in \mathcal{B} into disjoint subsets \mathcal{B}_1 and \mathcal{B}_2 which are then separately packed. For $DSA(5, 4, 2)$, the algorithm FF-BP partitions \mathcal{B} so that the subset \mathcal{B}_1 has density 2 ($m(\mathcal{B}_1) \leq 2$), and the subset \mathcal{B}_2 has density 3 ($m(\mathcal{B}_2) \leq 3$). Then the blocks in \mathcal{B}_1 are packed in 2 tracks using algorithm $DSA(2, 2, 2)$ and the blocks in \mathcal{B}_2 are packed in 3 tracks using algorithm $DSA(3, 3, 2)$.

Algorithm FF-BP goes as follows: blocks in \mathcal{B} are processed in increasing arrival times. Whenever a blocks B_j arrives, it is appended to \mathcal{B}_1 if $m_t(\mathcal{B}_1) + s_j \leq 2$. Otherwise, it is appended to \mathcal{B}_2 . (Details are given in Algorithm 2.)

The correctness of this procedure is shown as follows: First, $m(\mathcal{B}_1) \leq 2$ by construction, and so we only need to show that $m(\mathcal{B}_2) \leq 3$. Blocks of size 1 are trivially placed. Consider a block B_j of size 2 arriving at time t . Then, at most two tracks are occupied at time t (since total density is 4), namely, $m_t(\mathcal{B}_1) + m_t(\mathcal{B}_2) \leq 2$. If B_j cannot be appended to \mathcal{B}_1 , then $m_t(\mathcal{B}_1) \geq 1$ and so $m_t(\mathcal{B}_2) \leq 1$. Thus, $m_t(\mathcal{B}_2) \leq 3$ after the block B_j is appended to \mathcal{B}_2 .

Algorithm 2: DSA-Alg(5,4,2, \mathcal{B}): [FF-BP Algorithm]

1. **begin**
2. Sort the blocks in \mathcal{B} by arrival time;
3. $\mathcal{B}_1 \leftarrow \emptyset$; $m_1 = 2$;
4. $\mathcal{B}_2 \leftarrow \emptyset$; $m_2 = 3$;
5. **for each** block B_j **do**
6. $t \leftarrow a_i$;
7. **if** ($m_t(\mathcal{B}_1) + s_j \leq m_1$)
8. **then** $\mathcal{B}_1 \leftarrow \mathcal{B}_1 \cup \{B_j\}$;
9. **else** $\mathcal{B}_2 \leftarrow \mathcal{B}_2 \cup \{B_j\}$;
10. Apply DSA-Alg(2, 2, 2, \mathcal{B}_1);
11. Apply DSA-Alg(3, 3, 2, \mathcal{B}_2);
12. **end**

The algorithm FF-BP also solves $DSA(6, 5, 2)$ by partitioning the blocks into two subsets \mathcal{B}_1 and \mathcal{B}_2 having density 3 each. The proof is omitted here.

3.2 Basic Cases for $h = 3$

Theorem 5. *For the offline DSA problem, we have*

(a) $\mathcal{L}(3, 3) = 3$ and (b) $\mathcal{L}(4, 3) = 5$.

The proof of Theorem 5(a), namely, that $\mathcal{L}(3, 3) = 3$, is simple. We first assign all the blocks of size 3. Then between any two consecutive blocks of size 3, we have an instance of $DSA(3, 3, 2)$ that is feasible.

To prove Theorem 5(b), we first note that $DSA(4, 4, 3)$ is infeasible by Corollary 1. Next, we prove that $DSA(5, 4, 3)$ is feasible. The algorithm used is a generalization of the *First Fit with Channel Flip* procedure used in $DSA\text{-Alg}(3, 3, 2, \mathcal{B})$. We have two channels C_2 with 2 tracks and C_3 with 3 tracks. Without loss of generality, we assume that $C_2 = \{1, 2\}$ and $C_3 = \{3, 4, 5\}$.

Consider a block B_j arriving at time t . If $s_j = 1$, it is trivially assigned using first fit. If $s_j = 2$, if C_2 is free, B_j is assigned to C_2 otherwise, B_j is assigned to C_3 using $DSA\text{-Alg}(3, 3, 2)$ ³. If $s_j = 3$, and C_3 is free, B_j is assigned to C_3 . If C_3 is not free, then C_2 must be free and there is *exactly one* occupied track. If track 3 is free, we perform a channel flip at t so that the new $C_2 = \{4, 5\}$ and the new $C_3 = \{1, 2, 3\}$ is free and assign B_j to the new C_3 . (If track 3 is not free, we first invert the tracks in channel C_3 so that track 3 is now free.)

4 Approximation Algorithms for Special Cases

In this section, we show how the results for the small instances can be used to give improved approximation ratios, namely, $\mathcal{L}(h)$, for small h .

4.1 Algorithms Based on Multi-level Block Partitioning

We extend the block partitioning ideas used in $DSA\text{-Alg}(5, 4, 2)$ to a multi-level block partitioning algorithm called FF-MLBP. We illustrate the algorithm with the case $h = 2$. The block B_j in \mathcal{B} are considered in increasing arrival time. The algorithm proceeds in 2 phases as follows:

Phase 1: Let $n_1 = \lceil \frac{M}{3} \rceil$. We define n_1 level-1 subsets denoted by $\mathcal{B}_1^1, \mathcal{B}_2^1, \dots, \mathcal{B}_{n_1}^1$, each with maximum density $m_1 = 3$. Each block B_j is assigned using first fit to the first subset that can accommodate it. If B_j does not fit into any of these subsets, it is left unassigned (for Phase 2). (Note: We show later that all blocks of size 1 are assigned in Phase 1 and there are now only blocks of size 2 left.)

Phase 2: Now, define $n_2 = \lceil \frac{M}{6} \rceil$ level-2 subsets denoted by $\mathcal{B}_1^2, \mathcal{B}_2^2, \dots, \mathcal{B}_{n_2}^2$, each with maximum density $m_2 = 2$. It is trivial to assign the remaining blocks using first fit to these subsets.

Theorem 6. $DSA(4/3M + 4, M, 2)$ is feasible, namely, $\mathcal{L}(2) \leq \frac{4}{3}$.

³ Note that this assignment to channel C_3 may require an “internal” channel flip within the 3 tracks of C_3 .

Proof. We first prove the correctness of the algorithm. It is easy to see that all blocks of size 1 are assigned in Phase 1 since there are $n_1 = \lceil \frac{M}{3} \rceil$ subsets each with maximum density of 3. Hence, there are only blocks of size 2 left in Phase 2. Suppose that a block B of size 2 arriving at time t cannot be assigned to any one of the level-2 subsets (in Phase 2). This implies that at time t , each of these n_2 subsets must have density 2 (since there are no blocks of height 1). Furthermore, each of the level-1 subsets must have density at least 2 at time t (since block B was not assigned in Phase 1). Thus, the total density at time t (excluding block B) is $\geq 2n_1 + 2n_2 = 2 \lceil \frac{M}{3} \rceil + 2 \lceil \frac{M}{6} \rceil \geq M$. This contradicts the density bound M for all blocks. Thus, all the blocks are successfully assigned in Phase 2.

We now consider the number of tracks needed. Each of the n_1 level-1 subsets can be solved using DSA(3,3,2) and each of the n_2 level-2 subsets with DSA(2,2,2). Thus, the total number of tracks needed is $3n_1 + 2n_2 \leq \frac{4}{3}M + 4$. \square

This result improves the previous bound for $\mathcal{L}(2)$ from $3/2$ (by [MuBh99]) to $4/3$. This multi-level block partitioning approach can be generalized for $h = 3$ to give a bound of $11/6$ (≈ 1.833) for $\mathcal{L}(3)$ using a 3-phase block partitioning approach with $n_1 = \lceil \frac{M}{3} \rceil$, $n_2 = \lceil \frac{M}{6} \rceil$, $n_3 = \lceil \frac{M}{9} \rceil$, and with $m_1 = m_2 = m_3 = 3$. However, a superior result is described in the next section.

4.2 An 1.7-Approximation Algorithm for $h = 3$

In this section, we will present a special multi-level block partitioning algorithm for $h=3$ that gives a bound of 1.7 for $\mathcal{L}(3)$. We first describe a customized block partitioning algorithm (called FF-BP4) that assigns blocks to a packing area with 4 tracks. We divide the packing area into two channels C_1 and C_2 each with 2 contiguous tracks, say $C_1 = \{1, 2\}$ and $C_2 = \{3, 4\}$. A block of size 1 can be assigned to any free track. A block of size 2 can only be assigned to channel C_1 or C_2 (but not to the tracks 2 and 3 even if they are free). A block of size 3 can be assigned to tracks $\{1, 2, 3\}$ or $\{2, 3, 4\}$ if there is room (with one of the channels flipped if necessary). Any block that can not be assigned in this way is left unassigned.

Let \mathcal{B}_1 be the set of assigned blocks after running algorithm FF-BP4. We prove that $m_t(\mathcal{B}_1) \geq 2$ whenever a block B cannot be assigned at time t . If B is of size 1, then $m_t(\mathcal{B}_1) = 4$. If B is of size 2, then $m_t(\mathcal{B}_1) \geq 3$ except for the case when tracks $\{2, 3\}$ are free in which case $m_t(\mathcal{B}_1) \geq 2$. If B is of size 3, then $m_t(\mathcal{B}_1) \geq 2$, otherwise we always pack B successfully.

We are now ready to present the multi-level block partitioning algorithm for $h=3$. The algorithm is the same as the 3-phase FF-MLBP except that we use different algorithms to assign blocks to the subsets.

Phase 1: Define $n_1 = \lceil \frac{M}{4} \rceil$ level-1 subsets, with $m_1 = 4$ and assign blocks using algorithm FF-BP4. After this phase, only blocks of sizes 2 and 3 remain.

Phase 2: Define $n_2 = \lceil \frac{M}{10} \rceil$ level-2 subsets, with $m_2 = 6$ and assign blocks using the algorithm for DSA(6,6,{2,3}) from Theorem 6(d). All blocks of size 2 are assigned in this phase.

Phase 3: Define $n_3 = \lceil \frac{M}{30} \rceil$ level-3 subsets, with $m_3 = 3$ and assign blocks using first fit.

Theorem 7. *DSA($1.7M + 13$, M , 3) is feasible, namely, $\mathcal{L}(3) \leq 1.7$.*

Proof. We first prove the correctness of the algorithm. Suppose that a block B of size 2, arriving at time t cannot be assigned in Phase 2. This implies that at time t , each of the n_1 level-1 subsets must have density at least 2, and each of the n_2 level-2 subsets must have density at least 5. Thus, the total density at time t (excluding block B) is $\geq 2n_1 + 5n_2 = 2 \lceil \frac{M}{4} \rceil + 5 \lceil \frac{M}{10} \rceil \geq M$, which gives a contradiction. Thus, all blocks of size 2 are assigned in Phase 2.

Now, suppose that a block B of size 3, arriving at time t cannot be assigned in Phase 3. This implies that at time t , each of the n_1 level-1 subsets must have density at least 2, each of the n_2 level-2 subsets must have density at least 4, and each of the n_3 level-3 subsets must have density of 3. Thus, the total density at time t (excluding block B) is $\geq 2n_1 + 4n_2 + 3n_3 = 2 \lceil \frac{M}{4} \rceil + 4 \lceil \frac{M}{10} \rceil + 3 \lceil \frac{M}{30} \rceil \geq M$, which gives a contradiction. Thus, all blocks of size 3 are also assigned and the algorithm is correct.

The total number of tracks needed is $4n_1 + 6n_2 + 3n_3 \leq 1.7M + 13$. \square

References

- [GJ79] M.R. Garey and D.S. Johnson: Computers and Intractability – A guide to the Theory of NP-Completeness. Freeman, (1979).
- [Gerg96] J. Gergov: Approximation algorithms for dynamic storage allocation. European Symp. on Algorithms (ESA '96), Springer, LNCS 1136, (1996), 52–61.
- [Gerg99] J. Gergov: Algorithms for Compile-Time Memory Optimization. ACM-SIAM Symposium on Discrete Algorithms, (1999), 907–908.
- [Kier88] H. A. Kierstead: The linearity of first-fit coloring of interval graphs. SIAM J. Disc. Math., 1, (1988), 526–530.
- [Kier91] H. A. Kierstead: A polynomial time approximation algorithm for Dynamic Storage Allocation. Discrete Mathematics, 88, (1991), 231–237
- [Knut73] D.E. Knuth: Fundamental Algorithms: The art of computer programming. Volumn 1. Addison-Wesley Pub. (1973).
- [Li02] S.C. Li: Algorithms for Berth Allocation Problem, M.Sc Thesis, Department of Computer Science, National University of Singapore (2002).
- [LuNO74] M.G. Ludy, J. Naor, and A. Orda: Tight Bounds for Dynamic Storage Allocation. J. ACM, Vol 12, (1974), 491–499.
- [MuBh99] P. K. Murthy, S. S. Bhattacharyya: Approximation algorithms and heuristics for dynamic storage allocation problem. UMIACS TR-99-31, University of Maryland, College Park, (1999).
- [Robs74] J. M. Robson: Bounds for some functions concerning dynamic storage allocation. Journal of the ACM, Vol 21(3), (1974), 491–499.
- [Robs77] J. M. Robson: Worst base fragmentation of first fit and best fit storage allocation strategies. Computer Journal, Vol 20, (1977), 242–244.
- [Slus89] M. Slusarek: A Coloring Algorithm for Interval Graphs. Proc. 14th Math. Foundations of Computer Science, LNCS 379, Springer, (1989), 471–480.
- [WJNB95] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles: Dynamic storage allocation: A survey and critical review. Proceedings of International Workshop on Memory Management, LNCS 986, (1995).

***k*-Center Problems with Minimum Coverage**

Andrew Lim¹, Brian Rodrigues², Fan Wang¹, and Zhou Xu^{1,*}

¹ Dept Industrial Engineering and Engineering Management, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong
{iealim,fanwang,xuzhou}@ust.hk

² School of Business, Singapore Management University
br@smu.edu.sg

1 Introduction

The k -center problem is a well-known facility location problem and can be described as follows: Given a complete undirected graph $G = (V, E)$, a metric $d : V \times V \rightarrow \mathbb{R}_+$ and a positive integer k , we seek a subset $U \subseteq V$ of at most k centers which minimizes the maximum distances from points in V to U . Formally, the objective function is given by:

$$\min_{U \subseteq V, |U| \leq k} \max_{v \in V} \min_{r \in U} d(v, r).$$

As a typical example, we may want to set up k service centers (e.g., police stations, fire stations, hospitals, polling centers) and minimize the maximum distances between each client and these centers. The problem is known to be \mathcal{NP} -hard [2].

A factor ρ -approximation algorithm for a minimization problem is a polynomial time algorithm which guarantees a solution within at most ρ times the optimal cost. For the k -center problem, Hochbaum and Shmoys presented a factor 2-approximation algorithm and proved that no factor better than 2 can be achieved unless $\mathcal{P} = \mathcal{NP}$ [3]. Approximation algorithms for other k -center problems where vertex costs are considered or when vertex weights are used have been extensively studied [9]. More recently, Bar-Ilan, Kortsarz and Peleg investigated an interesting generalization of capacitated k -center problem where the number of clients for each center was restricted to a service capacity limit or maximum load [1]. Their work was improved in recent work by Khuller and Sussmann [7]. On the other hand, to ensure backup centers are available for clients, Krumke developed a "fault tolerant" k -center problem, where the objective was to minimize maximum distances as before, but where each client is required to be covered by a minimum number of centers [8]. Approximation algorithms for these problems were improved and extended in [6].

In these studies, no provision was made to ensure centers provide a minimum coverage of clients. In the fault tolerant problem, the client demand side of the problem is guaranteed coverage by a minimum number of centers (less than k), yet, on the supply side, there is no guarantee that each center services a minimum

* Corresponding Author

number of clients. In realistic applications however, such coverage is a common requirement. For example, in planning the location of hospitals, it would be expected that each hospital services a minimum number of neighborhoods. This would impose a balanced workload among hospitals and allow for economies of scale. Moreover, in cases when each center is equipped to provide a variety of services, a spread of clients covered is more likely to benefit service providers and clients alike. For example, where warehouses stock a variety of products, it would be beneficial if each services a spread of customers whose demands are more likely to include the range of products available. In this work, we address these provisions by extending the basic k -center problem to include a minimum coverage requirement. We allow coverage by different centers to overlap allowing clients to choose from a number of centers. In the problem, we minimize distances as in the basic k -center problem and require that every vertex in V is covered by one of the at most k selected centers in U . Further, each center in U must cover at least q vertices in V , where q is a non-negative integer, at most as large as $|V|$, which defines the minimum coverage for each center.

We call this a q -all-coverage k -center problem, with an objective function given by:

$$\min_{U \subseteq V, |U| \leq k} \max(\max_{v \in V} \min_{r \in U} d(v, r), \max_{r \in U} d_q(V, r)),$$

where $d_q(V, r)$ is the distance to r from its q^{th} closest vertex in V . Note that because $r \in V$, its closest vertex is r itself.

Further, two variations to this problem will be studied. The first is a q -coverage k -center problem, for which only vertices in $V - U$ are counted in the coverage of every center in U . Its objective function is:

$$\min_{U \subseteq V, |U| \leq k} \max(\max_{v \in V - U} \min_{r \in U} d(v, r), \max_{r \in U} d_q(V - U, r)),$$

where $d_q(V - U, r)$ is the distance of r from its q^{th} closest vertex in $V - U$.

The second is a q -coverage k -supplier problem for which V is partitioned into two disjoint subsets: S , a supplier set, and D , a demand set. The problem is then to find a subset U of at most k centers in S to minimize distances where not only is every demand point in D covered by a center in U , but every center in U must cover at least q demands in D . Here, the objective function is:

$$\min_{U \subseteq S, |U| \leq k} \max(\max_{v \in D} \min_{r \in U} d(v, r), \max_{r \in U} d_q(D, r)),$$

where that $d_q(D, r)$ is the distance of r from its q^{th} closest demands in D .

Additionally, these three problems can be generalized by the inclusion of vertex costs and vertex weights, as has been done for the basic k -center problem. To include costs, we define a cost $c(v)$ for each vertex v in V where we now require $\sum_{r \in U} c(r) \leq k$. This cost generalization is useful, for example, in the case of building centers where the cost for centers can vary and when there is a limited budget as is the case in practice.

To extend the problems by including weights, we take $w(v)$ be the weight of each vertex v in V so that the weighted distance to vertex v from vertex

u in V is $w(u, v) = d(u, v) \cdot w(v)$. For any vertex $v \in V$ and $X \subseteq V$, we let $w_q(X, v)$ to be the q^{th} closest weighted distance of v from the vertices in X . With this, the three variants can be generalized to weighted models by replacing distances d and d_q in the objective functions with the weighted distances w and w_q , respectively. Weighted distances can be useful, for example, when $1/w(v)$ is modelled to be the response speed of the center at vertex v , which then makes $w(u, v) = d(u, v) \cdot w(v)$ its response time.

Finally, by considering both vertex costs and vertex weights, we study the most general extensions for the three new problems.

Throughout this paper, OPT denotes the optimal value of the objective function. We assume that the complete graph $G = (V, E)$ is directed, where $V = \{v_1, \dots, v_n\}$ and $E = V \times V = \{e_1, \dots, e_m\}$ where $m = n^2$ where each vertex $v \in V$ has a self-loop $(v, v) \in E$ with distance $d(v, v) = 0$. For each edge $e_i \in E$, let $d(e_i)$ and $w(e_i)$ denote its distance and its weighted distance, respectively. A vertex v is said to *dominate* a vertex u , if and only if v is equivalent to u ($v = u$) or v is adjacent to u and we denote the number of vertices dominated by v in G by $\deg^+(v)$. For each vertex v in the undirected graph H , $\deg(v)$ is its *degree*, i.e. the number of adjacent edges including the possible self-loop (v, v) , and for any undirected graph H , $I(H)$ denotes a *maximal independent set* [2], in which no two different vertices share an edge in H and no vertex outside $I(H)$ can be included while preserving its independence.

We present approximation algorithms for the three problems considered in this paper and their generalizations. Our methods extend from the threshold technique used for the basic k -center problem [4], and are designed to address the new minimum coverage constraints included.

2 Main Results

Our main results are summarized as follows.

	Basic	Weights	Costs	Weights + Costs
q -All-Coverage K -center	2^\dagger	3	3^\ddagger	$2\beta + 1$
q -Coverage K -center	2^\dagger	4	4	$3\beta + 1$
q -Coverage K -supplier	3^\dagger	3^\dagger	3^\dagger	$2\beta + 1$

We use \dagger to indicate the best possible approximation factors have been achieved, which are shown to be 2, 2 and 3 for the three problems, respectively, unless $\mathcal{P} = \mathcal{NP}$. These optimal results include the basic cases of all the three problems considered, and the weight and the cost generalizations of the q -coverage k -supplier problem. Moreover, for the weight and the cost generalizations of the other two problems, approximation algorithms are provided with constant factors, all of which are close to their best possible approximation factor of 2. Especially, for the cost generalization of the q -all-coverage k -center problem indicated by \ddagger in the table, a 3-approximation algorithm is achieved which matches the best known approximation factor for the cost generalization of the classical k -center problem [4]. Further to this, the approximation algorithms for the cost generalizations of the three problems can be extended to solve their weight plus

cost generalizations. Let β denote the ratio between the maximum value and the minimum value of weights. Their approximation factors are consistent with those of their cost generalizations, which hold when $\beta = 1$.

3 q -All-Coverage k -Center Problems

The following hardness result for the q -all-coverage k -center problem can be proved by extending the reduction from the *Domination Set* problem [2] used for the classical k -center problem [5]. Details of the proof are omitted here but described in [10], for the shortage of space.

Theorem 1. *Given any fixed non-negative integer q , there is no polynomial-time $(2 - \varepsilon)$ -approximation algorithm for the q -all-coverage k -center problem, unless $\mathcal{NP} = \mathcal{P}$.*

The best possible approximation factor of 2 can be achieved by Algorithm 1. We first sort edges in E by order of non-decreasing distances, i.e., $d(e_1) \leq d(e_2) \leq \dots \leq d(e_m)$. Let $G_i = (V, E_i)$ where $E_i = \{e_1, \dots, e_i\}$ for $1 \leq i \leq m$. Thus, if G_i has a set U of at most k vertices that dominate all vertices in G_i , and each vertex of U dominates at least q vertices (including itself) in G_i , then U provides at most k centers to the problem with at most $d(e_i)$ distance. Let i^* denote the smallest such index. So $d(e_{i^*}) = OPT$ is the optimal distance.

To find a lower bound for OPT , construct an undirected graph H_i . H_i contains an edge (u, v) where $u, v \in V$ might be equal if and only if there exists a vertex $r \in V$ with $\deg^+(r) \geq q$ and both (u, r) and (v, r) are in G_i . It is clear that the self loop (v, v) remains in H_i for each $v \in V$, and that if $(v, u) \in G_i$ then $(v, u) \in H_i$ since (v, v) and (u, v) are in G_i . As any two vertices dominated by the same vertex in G_i are adjacent in H_i , H_{i^*} satisfies the following: (1) for each vertex $v \in V$, $\deg(v) \geq q$ in H_{i^*} , including its self-loop; (2) the size of its maximal independent set $|I(H_{i^*})| \leq k$. Accordingly, suppose that the threshold j is the minimum index i leading H_i to satisfy the above two conditions, then we have $j \leq i^*$, which gives $d(e_j) \leq OPT$.

Finally, selecting vertices in the maximal independent set H_j , we have $|I(H_j)| \leq k$. So, centers in $I(H_j)$ dominate all vertices of V in H_j , and each $v \in I(H_j)$ dominates at least $\deg(v) \geq q$ vertices (including itself) of V in H_j . By the triangle inequalities, we know $d(u, v) \leq 2d(e_j) \leq 2 \cdot OPT$, for every (u, v) in H_j . So the set U gives at most k centers with at most $2 \cdot OPT$ distance, which establishes the following theorem.

Algorithm 1 (Basic q -All-Coverage k -Center)

- 1: Sort edges so that $d(e_1) \leq d(e_2) \leq \dots \leq d(e_m)$, and construct H_1, H_2, \dots, H_m ;
 - 2: Compute a maximal independent set, $I(H_i)$, in each graph H_i , where $1 \leq i \leq m$;
 - 3: Find threshold j , denoting the smallest index i , such that $|I(H_i)| \leq k$, and for each $v \in V$, $\deg(v) \geq q$ in H_i ;
 - 4: Return $I(H_j)$.
-

Theorem 2. *Algorithm 1 achieves an approximation factor of 2 for the q -all-coverage k -center problem.*

3.1 Any q with Weights

For the weighted case of the q -all-coverage k -center problem, a 3-approximation algorithm follows. Firstly, sort edges by nondecreasing weighted distances, i.e., $w(e_1) \leq w(e_2) \leq \dots \leq w(e_m)$ and let $G_i = (V, E_i)$ where $E_i = \{e_1, \dots, e_i\}$. The graph H_i for G_i contains an edge (u, v) where $u, v \in V$ might be equal if and only if there exists a vertex r which has both (u, r) and (v, r) in G_i , which implies $w(u, r) \leq w(e_i)$ and $w(v, r) \leq w(e_i)$. To bound the optimum weighted distance (OPT), find the minimum index i and threshold j so that the degree of each vertex in H_i is at least q and the size of its maximal independent set is $|I(H_i)| \leq k$. This ensures $w(e_j) \leq OPT$. Finally, consider each vertex $v \in V$. Among all $u \in V$ with $w(v, u) \leq w(e_i)$, let $g_i(v)$ denote the vertex having the smallest weight, i.e., the least weighted neighbor of v in G_i . Shifting every $v \in I(H_j)$ to $g_j(v)$, we obtain the set U which guarantees an approximation factor of 3 given by the following theorem.

Theorem 3. *An approximation factor of 3 can be achieved polynomially for the weighted q -all-coverage k -center problem.*

Proof. Firstly, by $|I(H_j)| \leq k$ and $U = \{g_j(v) | v \in I(H_j)\}$, we have $|U| \leq k$. Next, for any vertex $u \in V$, there must exist v in $I(H_j)$ with (u, v) in H_j , which gives a vertex $r \in V$ with both (u, r) and (v, r) in G_j . Hence, $w(u, r) \leq w(e_j)$ and $w(v, r) \leq w(e_j)$. Since $w(g_j(v)) \leq w(r)$ and $w(v, g_j(v)) \leq w(e_j)$, u is covered by $g_j(v) \in U$ within $w(u, g_j(v)) \leq (d(u, r) + d(v, r) + d(v, g_j(v)))w(g_j(v)) \leq 3w(e_j)$. Furthermore, the degree of any vertex $v \in I(H_j)$ is at least q , which implies at least q vertices like u , equivalent or adjacent to v in H_j , can be covered by $g_j(v) \in U$ within $3w(e_j)$. Since $w(e_j) \leq OPT$, the approximation factor is 3. \square

3.2 Any q with Weights and Costs

We now give a $(2\beta+1)$ -approximation algorithm for the most general case where vertices have both weights and costs. If only cost is considered, a 3-approximation can be achieved where $\beta = 1$.

The algorithm here is similar to that for the q -all coverage k -center problem, except that a new set U_i is constructed by shifting each $v \in I(H_i)$ to $s_i(v)$, where $s_i(v)$ is the vertex who has the lowest cost among all $u \in V$ with $w(v, u) \leq w(e_i)$. Hence $s_i(v)$ is called the cheapest neighbor of v in G_i and we take $U_i = \{s_i(v) | v \in I(H_i)\}$ and $c(U_i)$ to denote the total costs of vertices in U_i . Because no two vertices in $I(H_i)$ are dominated by a common vertex in G_i , the index i^* with $w(e_{i^*}) = OPT$ leads H_{i^*} to satisfy the following: (1) for each vertex $v \in V$, $\deg(v) \geq q$ in H_{i^*} , including its self-loop; (2) $c(U_{i^*}) \leq k$.

Finding the threshold j to be the minimum index i which causes H_i to satisfy the above two conditions, we have $j \leq i^*$ and $w(e_j) \leq OPT$. Furthermore, we

will prove that the U_j provides at most k cost centers ensuring an approximation factor of $(2\beta + 1)$ in the following.

Theorem 4. *An approximation factor of $(2\beta + 1)$ can be achieved polynomially for the weighted and cost q -all-coverage k -center problem.*

Proof. Because $c(U_j) \leq k$ and $w(e_j) \leq OPT$ have been shown, we need only show that the objective function distance given by U_j is at most $(2\beta + 1)w(e_j)$. On one hand, for any vertex $u \in V$, there exists a vertex $v \in I(H_j)$ adjacent to u in H_j . This implies there is a vertex $r \in V$ with both $w(u, r) \leq w(e_j)$ and $w(v, r) \leq w(e_j)$. Since $w(r) \leq \beta w(s_j(v))$ and $w(v, s_j(v)) \leq w(e_j)$, we know that v is covered by $s_j(v) \in U_j$ within $w(u, s_j(v)) \leq (d(u, r) + d(v, r) + d(v, s_j(v)))w(s_j(v)) \leq (2\beta + 1)w(e_j)$. On the other hand, because the degree of any vertex $u \in I(H_j)$ is at least q , there are at least q vertices like u , equivalent or adjacent to v in H_j , covered by $s_j(v) \in U_j$ within at most $(2\beta + 1)w(e_j)$ weighted distance. Since $w(e_j) \leq OPT$, the approximation factor is $(2\beta + 1)$. \square

4 q -Coverage k -Center Problems

Compared with the q -all-coverage k -center problem, the q -coverage k -center problem has an additional stipulation: for each selected center v , the at least q vertices covered by v should be outside the set of selected centers.

To determine its hardness, we provide the following theorem, which can be shown by a modified reduction used for Theorem 1. Details of the proof are omitted here but described in [10], for the shortage of space.

Theorem 5. *Given any fixed non-negative integer q , there is no polynomial-time $(2 - \varepsilon)$ -approximation algorithm for the q -coverage k -center problem, unless $\mathcal{NP} = \mathcal{P}$.*

The best possible approximation factor of 2 can be achieved for the q -coverage k -center problem in a similar way as the algorithm for the q -all-coverage k -center problem. The only difference is that the threshold j , found here, must cause the degree $\deg(v)$ to be at least $q + 1$ in H_j instead of q for each vertex $v \in V$, since self-loops might exist but each center should be adjacent to q vertices other than itself. The approximation factor of 2 is proved by the following theorem.

Theorem 6. *An approximation factor of 2 can be achieved polynomially for the q -coverage k -center problem.*

Proof. By the same analysis for Theorem 2, we know that $d(e_j) \leq OPT$, and that $I(H_j)$ provides at most k centers which cover all the vertices within at most $2d(e_j)$. Since each vertex $v \in I(H_j)$ is adjacent to at least q vertices other than itself in H_j , to prove its q -coverage within $2d(e_j)$ we need only show that no two vertices in $I(H_j)$ are adjacent to each other in H_j . This is obvious, since $I(H_j)$ is an independent set of H_j . Since $2d(e_j) \leq 2 \cdot OPT$, the approximation factor is 2. \square

Algorithm 2 (weighted q -Coverage k -Center)

- 1: Sort edges so that $w(e_1) \leq w(e_2) \leq \dots \leq w(e_m)$;
 - 2: For each $1 \leq i \leq m$, let $Q_i = \{v \mid \deg^+(v) \geq q + 1\}$;
 - 3: Construct graphs, P_1, P_2, \dots, P_m ;
 - 4: Compute a maximal independent set, $I(P_i)$ for each graph P_i where $1 \leq i \leq m$;
 - 5: Find the threshold j , denote the smallest index i , such that Q_i dominates all vertices of V in G_i , and $|I(P_i)| \leq k$;
 - 6: For each vertex $v \in I(P_j)$, let $p(v)$ denote the lowest weighted vertex among all vertices $u \in Q_j$ with an edge (v, u) in G_j ;
 - 7: Shift vertices in $I(P_j)$ by $U' = \{p(v) \mid v \in I(P_j)\}$;
 - 8: Construct $H' = (U', E')$ from G_j , where for any two vertices u and v in $I(P_j)$, an edge $(p(u), p(v)) \in E'$ if and only if either $(p(v), p(u))$ or $(p(u), p(v))$ is in G_j ;
 - 9: Call Algorithm 3 to obtain $I(H')$, a maximal independent set of H' , insuring that for any vertex $u \in U' - I(H')$, there exists a vertex $v \in I(H')$ with $(u, v) \in E'$ and $w(v) \leq w(u)$;
 - 10: Return $I(H')$.
-

4.1 Any q with Weights

The weighted case of the q -coverage k -center problem can be solved by Algorithm 2, which is more intricate than previous algorithms and can be described as follows.

First, after sorting the m edges, an undirected graph P_i , instead of H_i , is constructed from G_i for $1 \leq i \leq m$. The construction is as follows. Let Q_i be the subset of $v \in V$ with $\deg^+(v) \geq q + 1$. For any $u, v \in V$ where u and v might be equal, an edge (u, v) is in P_i if and only if there exists $r \in Q_i$ so that both (u, r) and (v, r) are in G_i . Consider the index i^* where $w(e_{i^*}) = OPT$. Because each selected center must dominate at least q vertices other than itself, and no two vertices in $I(P_{i^*})$ are dominated by the same vertex in G_{i^*} , we observe that: (1) each vertex of V is dominated by at least one vertex in Q_{i^*} ; (2) the size of $I(P_{i^*})$ must be at least as large as k , i.e. $|I(P_{i^*})| \leq k$. Accordingly, define the threshold j to be the smallest index i , such that Q_i dominates all vertices of V , and $|I(P_i)| \leq k$. The two observations above imply $w(e_j) \leq OPT$.

Second, shift vertices in $I(P_j)$ as follows. For each vertex $v \in I(P_j)$, let $p(v)$ denote the smallest weighted vertex, among all $u \in Q_j$ with an edge (v, u) in G_j . This gives $U' = \{p(v) \mid v \in I(P_j)\}$.

Now, consider an undirected graph, $H' = (U', E')$, where, for any two vertices u and v in $I(P_j)$, an edge $(p(u), p(v)) \in E'$ if and only if either $(p(v), p(u))$ or $(p(u), p(v))$ is in G_j . Its maximal independent set, denoted by $I(H')$, can be obtained greedily by Algorithm 3. It is easily seen that for any vertex $u \in U' - I(H')$, there exists a vertex $v \in I(H')$ with $(u, v) \in E'$ and $w(v) \leq w(u)$, where v could be the vertex that marks u in Algorithm 3.

Now, we prove that $I(H')$ provides at most k centers ensuring an 4-approximation factor to establish the following theorem.

Theorem 7. *Algorithm 2 achieves an approximation factor of 4 for the weighted q -coverage k -center problem.*

Proof. Noting $w(e_j) \leq OPT$ and $|I(H')| \leq |U'| \leq |I(P_j)| \leq k$, we need only prove the following two facts: (1) each $p(v) \in I(H')$ covers at least q vertices $u \in V - I(H')$ within $w(u, p(v)) \leq 4w(e_j)$, where $v \in I(P_j)$; (2) each $u \in V - I(H')$

Algorithm 3 (Maximal Independent Set of $H' = (U', E')$ with Weights w)

```

1:  $U \leftarrow \emptyset$ ;
2: while  $U' \neq \emptyset$  do
3:   Choose the vertex  $v$ , which has the smallest weight  $w(u)$  among all  $u \in U'$ ;
4:    $U \leftarrow U + \{v\}$  and  $U' \leftarrow U' - \{v\}$ ;
5:   Mark all the vertices  $u \in U'$  adjacent to  $v$ , i.e.  $(u, v) \in E'$ , by  $U' \leftarrow U' - \{u\}$ ;
6: end while
7: Return  $U$  as the maximal independent set of  $H'$ .

```

is covered by a certain vertex $p(v) \in I(H')$ within $w(u, p(v)) \leq 4w(e_j)$, where $v \in I(P_j)$. On one hand, consider each $p(v) \in I(H')$, where $v \in I(P_j)$. Because $I(H') \subseteq U' \subseteq Q_j$, we know $p(v) \in Q_j$, and so, there exist at least q vertices, other than $p(v)$, which are dominated by $p(v)$ in G_j . Moreover, each vertex u of these q vertices is not in $I(H')$, because otherwise, the edge $(u, p(v))$ in G_j implies an edge $(u, p(v))$ in H' , contradicting to the independence of $I(H')$. Note that $w(u, p(v)) \leq w(e_j) \leq 4w(e_j)$. The fact 1 is proved.

On the other hand, consider each vertex $u \in V - I(H')$. Because $I(P_j)$ is a maximal independent set of P_j , there exists a vertex $t_1 \in I(P_j)$ with an edge (u, t_1) in P_j . (Note that if u is in $I(P_j)$, a self loop (u, u) must be in P_j because all vertices of V are dominated by Q_j). Thus, we know that $p(t_1)$ is in H' . Since $I(H')$ is a maximal independent set of H' , there exists a vertex $p(t_2) \in I(H')$ for $t_2 \in I(P_j)$, with $w(p(t_2)) \leq w(p(t_1))$ and an edge $(p(t_1), p(t_2))$ in H' . So $w(p(t_1), p(t_2)) \leq w(p(t_2), p(t_1))$, implying $(p(t_1), p(t_2))$ is in G_j . Because (u, t_1) is in P_j , there exists a vertex $a \in Q_j$ dominating both u and t_1 in G_j , leading $w(p(t_1)) \leq w(a)$. Noting that weighted distances of (u, a) , (t_1, a) , $(t_1, p(t_1))$, and $(p(t_1), p(t_2))$ are all at most $w(e_j)$, we have $w(u, p(t_2)) \leq (d(u, a) + d(t_1, a) + d(t_1, p(t_1)) + d(p(t_1), p(t_2)))w(p(t_2)) \leq 4w(e_j)$. This proves the fact 2 and completes the proof. \square

4.2 Any q with Weights and Costs

The basic idea employed to solve this problem is to combine and modify algorithms for the q -all-coverage and the q -coverage k -center problems. Here, we construct H_1, \dots, H_m first and sort edges e_1, \dots, e_m by their nondecreasing weighted distances.

However, to find the threshold j we need a new approach. For $1 \leq i \leq m$, an undirected graph H'_i is generated in the following manner: For any two vertices $u, v \in V$, the edge (u, v) is in H'_i , if and only if there exists a vertex $r \in V$, such that either (u, r) is in G_i and (v, r) is in H_i , or (v, r) is in G_i and (u, r) is in H_i . We then compute $I(H'_i)$, a maximal independent set of H'_i , and shift each vertex $v \in I(H'_i)$ to its lowest cost neighbor $s_i(v)$ in G_i ; this forms the set $U'_i = \{s_i(v) | v \in I(H'_i)\}$.

Now, we find the threshold j , which is the minimal index i , giving $\deg(v) \geq q + 1$ in H_i for each vertex $v \in V$ and $c(U'_i) \leq k$, where $c(U'_i)$ denotes the total cost of vertices in U'_i . Observing that H'_i is a subgraph of H_i , we know $I(H'_i)$ is also an independent set of H_i . By similar arguments for algorithms of the q -all-coverage and the q -coverage k -center problems, we derive $w(e_j) \leq OPT$.

Furthering, we can show that U'_j gives at most k cost centers within at most $(3\beta + 1) \cdot OPT$ weighted distance. (See [10] for details.)

Theorem 8. *An approximation factor of $3\beta + 1$ can be achieved polynomially for the weighted and cost q -coverage k -center problem.*

In addition, for the q -coverage k -center problem with cost only, an approximation factor of 4 can be obtained by $\beta = 1$.

5 q -Coverage k -Supplier Problems

The q -coverage k -supplier problem partitions the vertex set V into the supplier set S and the demand set D that are disjoint. Hence, at most k centers need be selected from S , to minimize the distance within which all the vertices in set D are covered by centers each of which must cover at least q suppliers in D . In order to determine its hardness, we present the following theorem which can be proved by a reduction of *Minimum Cover* problem [2]. Details of the proof are omitted here but described in [10], for the shortage of space.

Theorem 9. *Given any fixed non-negative integer q , there is no polynomial-time $(3 - \varepsilon)$ -approximation algorithm for the q -coverage k -center problem, unless $\mathcal{NP} = \mathcal{P}$.*

The best possible approximation factor of 3 can be achieved for the q -coverage k -supplier problem, even for its weighted extension and its cost extension. In the rest of this section, we first provide the weighted case a 3-approximation algorithm that is applicable for the basic case by specifying $w(u) = 1$ for each supplier $u \in S$. Then, we provide the weighted and cost case a $(2\beta + 1)$ -approximation algorithm that ensures a factor of 3 for the cost only case with $\beta = 1$.

5.1 Any q with Weights

The approximation approach follows. As before, edges are sorted non-decreasingly, i.e., $w(e_1) \leq w(e_2) \leq \dots \leq w(e_m)$. We have subgraphs G_1, G_2, \dots, G_m , where $G_i = (V, E_i)$, $V = S \cup D$ and $E_i = \{e_1, \dots, e_i\}$. To obtain the threshold index, a new graph L_i is constructed on the demand set D for each G_i as follows. For each two demands $u, v \in D$ where u may equal to v , an edge (u, v) is in L_i if and only if there exists a supplier $r \in S$ with both (u, r) and (v, r) in G_i . Hence, self-loops of all the vertices in V are still in L_i . Let $I(L_i)$ denote a maximal independent set of L_i . We find j to be the threshold index, which is the smallest index i , with $\deg(v) \geq q$ in L_i for $v \in D$, and $|I(L_i)| \leq k$. Since i^* , the edge index of the optimal solution satisfies the above two conditions, and no two demands in $I(L_i)$ have edges from the same supplier in G_i for $1 \leq i \leq m$, we have $j \leq i^*$ leading to $w(e_j) \leq OPT$.

We shift each demand $v \in I(L_j)$ to its cheapest supplier $g_j(v)$ with the lowest weight among suppliers having an edge from v in G_j . This forms the center set $U = \{g_j(v) | v \in I(L_j)\}$, which provides at most k centers with at most a $3 \cdot OPT$ weighted distance. To see this, we prove the following theorem.

Theorem 10. *An approximation factor of 3 can be achieved polynomially for the weighted q -coverage k -center problem.*

Proof. Note $|U| \leq |I(L_j)| \leq k$ and $w(e_j) \leq OPT$. To obtain the approximation factor of 3, we need only show the following two facts: (1) each demand $u \in D$ is covered by a vertex in U within at most $3w(e_j)$; (2) each supplier $g_j(v) \in U$, where $v \in I(L_j)$, covers at least q demands in D within at most $3w(e_j)$. See [10] for details. \square

5.2 Any q with Weights and Costs

An approximation factor of $(2\beta+1)$ can be obtained for the q -coverage k -supplier problem with weights and costs. When $\beta = 1$, it ensures an approximation factor of 3 for the cost only case. Compared with the algorithm for the weighted q -coverage k supplier problem, it is changed as follows.

After finding $I(L_i)$ for $1 \leq i \leq m$, we shift each demand $v \in I(L_i)$ to its cheapest supplier $s_i(v)$ with the lowest cost among all the suppliers having an edge from v in G_i . This forms $U_i = \{s_i(v) | v \in I(L_i)\}$. The threshold index j is the smallest index i , giving $\deg(v) \geq q$ in L_i for $v \in D$ and the total cost of vertices in U_i , $c(U_i)$, is at most k . Since no two demands in $I(L_i)$ have edges from the same supplier in G_i for $1 \leq i \leq m$, we have $w(e_j) \leq OPT$.

Now, we can prove that U_j have at most k cost centers within at most $(2\beta+1) \cdot OPT$ weighted distance to establish the following theorem, by similar arguments in Theorem 10. (See [10] for details.)

Theorem 11. *An approximation factor of $(2\beta+1)$ can be achieved polynomially for the weighted and cost q -coverage k -center problem.*

6 Conclusion

We studied a new k -center problem which ensures minimum coverage of clients by centers. It is motivated by the need to balance services provided by centers while allowing centers to be utilized fully. We considered three variants of the problem by examining their in-approximation hardness and approximation algorithms.

References

1. Judit Bar-Ilan, Guy Kortsarz, and David Peleg. How to allocate network centers. *Journal of Algorithms*, 15(3):385–415, November 1993.
2. M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
3. D. Hochbaum and D. B. Shmoys. A best possible heuristic for the k -center problem. *Mathematics of Operations Research*, 10:180–184, 1985.
4. D. Hochbaum and D. B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM*, 33:533–550, 1986.

5. W. Hsu and G. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1:209–216, 1979.
6. Samir Khuller, Robert Pless, and Yoram J. Sussmann. Fault tolerant k -center problems. *Theoretical Computer Science*, 242(1–2):237–245, 2000.
7. Samir Khuller and Yoram J. Sussmann. The capacitated k -center problem. *SIAM Journal on Discrete Mathematics*, 13(3):403–418, 2000.
8. S.O. Krumke. On a generalization of the p -center problem. *Information Processing Letters*, 56(2):67–71, 1995.
9. J. Plesnik. A heuristic for the p -center problem in graphs. *Discrete Applied Mathematics*, 17:263–268, 1987.
10. Zhou Xu, Andrew Lim, Brian Rodrigues, and Fan Wang. Coverage commitment k -center problems (long version). <http://www.comp.nus.edu.sg/~judge/doc.html>, unpublished, Sept. 2003.

On the Extensions of Solovay-Reducibility

Xizhong Zheng^{1,2,*} and Robert Rettinger³

¹ Department of Computer Science, Jiangsu University, Zhenjiang 212013, China

² Theoretische Informatik, BTU Cottbus, D-03044 Cottbus, Germany

zheng@informatik.tu-cottbus.de

³ Theoretische Informatik II, FernUniversität Hagen, D-58084 Hagen, Germany

robert.reettinger@fernuni-hagen.de

Abstract. A c.e. real x is Solovay reducible (S-reducible) to another c.e. real y if y is at least as difficult to be approximated as x . In this case, y is at least as random as x . Thus, the S-reducibility classifies relative randomness of c.e. reals such that the c.e. random reals are complete in the class of c.e. reals under the S-reducibility. In this paper we investigate extensions of the S-reducibility outside the c.e. reals. We show that the straightforward extension does not behave satisfactorily. Then we introduce two new extensions which coincide with the S-reducibility on the c.e. reals and behave reasonably outside the c.e. reals. Both of these extensions imply the rH-reducibility of Downey, Hirschfeldt and LaForte [6]. At last we show that even under the rH-reducibility the computably approximable random reals cannot be characterized as complete elements of this reduction.

1 Introduction

According to Turing [13], a real number $x \in [0, 1]$ is *computable* if it has a computable binary (or decimal) expansion, i.e., $x = x_A := \sum_{n \in A} 2^{-(n+1)}$ for a computable set $A \subseteq \mathbb{N}$. Thus, the computable reals have the simplest binary expansions. On the other hand, the binary expansion of a random real should be the most complicated, i.e., random. Here the randomness of a binary sequence α can be described by the “typicalness” that it does not belong to any “small” class according to Martin-Löf [9], or equivalently, by “chaoticness” which means that the sequence is disordered and complex such that the simplest way to describe it is to write all its members. That is, $H(\alpha \upharpoonright n) \geq n - c$ for some constant c and all n , where $H(\alpha \upharpoonright n)$ is the prefix-free Kolmogorov complexity of the initial segment $\alpha \upharpoonright n$ of length n (see, e.g. [5] for more details).

Relative computability is studied by means of various reducibilities. Similarly, Solovay’s domination relation of [12] compares the relative randomness of the c.e. (computably enumerable) reals, where c.e. reals are limits of increasing computable sequences of rationals. According to Solovay [12], a c.e. real y *dominates* another c.e. real x (or more popularly, x is *Solovay reducible* or *S-reducible*

* Corresponding address: Theoretische Informatik, BTU Cottbus, D-03044 Cottbus, Germany.

to y), if y is at least as difficult to be approximated as x . More precisely, we have the following definition, where $\mathcal{CS}(x)$ ($\mathcal{ICS}(x)$) is the class of computable (increasing) sequences of rationals converging to x .

Definition 1 (Solovay [12]). A c.e. real x is *Solovay reducible* to a c.e. real y ($x \leq_S y$) if there are $(x_s) \in \mathcal{ICS}(x)$, $(y_s) \in \mathcal{ICS}(y)$ and a constant c such that

$$(\forall s \in \mathbb{N}) (x - x_s \leq c(y - y_s)). \quad (1)$$

Solovay shows in [12] that S-reducibility has the following important *Solovay property* that, for any c.e. reals x and y

$$x \leq_S y \implies (\exists c)(\forall n)(H(x \upharpoonright n) \leq H(y \upharpoonright n) + c). \quad (2)$$

In other words, if $x \leq_S y$, then x cannot be not more random than y . Thus, Solovay reducibility classifies relative randomness of c.e. reals. Since Solovay reducibility is reflexive and transitive, the induced equivalent classes are defined as Solovay degrees (S-degrees). Downey, Hirschfeldt, and Nies [7] showed that the S-degrees of all c.e. reals form a dense distributive uppersemilattice, where the natural join operation is induced simply by addition. As expected, the least S-degree consists of all computable reals. More interestingly, the class of all c.e. random reals forms the largest S-degree among the c.e. S-degrees. This is proven by a series of works. At first, Chaitin [4] introduced Ω reals as halting probabilities of universal self-delimited Turing machines and showed that every Ω real is c.e. random. Solovay [12] called a c.e. real Ω -like if it is complete under the Solovay reducibility on the c.e. reals, i.e., every c.e. real is S-reducible to it. Then he showed that every Ω real is Ω -like and every Ω -like real is c.e. random. Next, Calude, Hertling, Khoussainov and Wang [3], showed that every Ω -like real is actually an Ω real. Finally, Kučera and Slaman [8] closed this circle by showing that any c.e. random real is Ω -like. Therefore, the notions Ω real, Ω -like real and c.e. random real are equivalent.

It is well known that Solovay reducibility is a good tool to study relative randomness of c.e. reals. Therefore, it is interesting how this reducibility can be extended to larger classes of reals. To this end, let's first recall some extensions of the class of c.e. reals.

As the limits of increasing computable sequences of rationals, c.e. reals are also called *left computable* because they are computably approximable from the left side on the real axes. Symmetrically, *right computable* reals are the limits of decreasing computable sequences of rationals. Left and right computable reals are called *semi-computable*. The classes of computable, left computable, right computable and semi-computable reals are denoted by **EC**, **LC**, **RC** and **SC**, respectively. A first interesting extension of c.e. reals is the class of weakly computable reals by Ambos-Spies, Weihrauch and Zheng [1]. A real x is *weakly computable* if it is the difference of two c.e. reals. Thus, weakly computable reals are also called d-c.e (difference of c.e. reals). It is shown in [1], that x is weakly computable iff there is an $(x_s) \in \mathcal{CS}(x)$ such that $\sum_{s \in \mathbb{N}} |x_s - x_{s+1}|$ is bounded and the class of weakly computable reals (denoted by **WC**) is

the arithmetical closure of all c.e. reals. More generally, the limit of a computable sequence of rationals (without any restriction) is called *computably approximable* (c.a., for short) and **CA** denotes the class of all c.a. reals. Then we have **EC** \subsetneq **LC** \subsetneq **SC** \subsetneq **WC** \subsetneq **CA** as shown in [1].

Another kind of extension of c.e. reals is motivated by the notion of monotone convergence of Calude and Hertling [2] and is introduced by the authors [10]. Let $h : \mathbb{N} \rightarrow \mathbb{Q}$ be a function. A sequence $(x_s) \in \mathcal{CS}(x)$ converges to x *h-monotonically* if $|x - x_t| \leq h(s)|x - x_s|$ for all $s \leq t$. A real x is called *h-monotonically computable* (*h-mc*, for short) if there is an $(x_s) \in \mathcal{CS}(x)$ which converges to x *h-monotonically*. x is called *monotonically computable* (*mc* for short) or *ω -monotonically computable* (*ω -mc*, for short) if it is *h-mc* for a constant function $h \equiv c$ or a computable function h , respectively. We denote by **MC** and ω -**MC** the classes of *mc* and *ω -mc* reals, respectively. It is shown in [11, 10] that **SC** \subsetneq **MC** \subsetneq ω -**MC** \subsetneq **CA** and **MC** \subsetneq **WC**, while the classes **WC** and ω -**MC** are incomparable.

As the first extension of the S-reducibility, the S2-reducibility is obtained directly by replacing the increasing computable sequence in Definition 1 by a general computable sequence. This extends the S-reducibility to the class of c.a. reals. The S2-reducibility coincides with the S-reducibility on c.e. reals and works also satisfactorily on the class **MC**. However, it is quite ill-behaved outside the class **MC** (Some of these ill-behaviors of the straightforward extension of the S-reducibility outside the c.e. reals were also pointed out by Downey, Hirschfeldt and LaForte in [6]). For example, S2-reducibility is not transitive on **WC** and some weakly computable real is not above a computable real, which contradicts the intuition of a reducibility respecting relative randomness.

As alternatives, Downey, Hirschfeldt and LaForte introduced in [6] two new reducibilities, the *strong weak truth table reducibility* (sw-reducibility) and the *relative H reducibility* (rH-reducibility). Both are defined for all reals and have the Solovay property. Nevertheless, the sw-reducibility is not comparable with the S-reducibility on the c.e. reals (as shown in [6]) and there is no maximal sw-degree of c.e. reals as shown by Yu and Ding [14]. The rH-reducibility, on the other hand, has some nice properties. But it does not coincide with the S-reducibility on the c.e. reals. In this sense, both sw-reducibility and rH-reducibility are not really extensions of the S-reducibility.

In this paper, two natural extensions of the S-reducibility are introduced which coincide with the S-reducibility on the c.e. reals and behave reasonably well outside the c.e. reals. The first one (the S2a-reducibility) works on the class **CA** which is obtained by introducing a small tolerance in the definition of S2-reducibility. Namely, instead of an inequation like $|x_s - x| \leq c|y_s - y|$, we require $|x_s - x| \leq c(|y_s - y| + 2^{-s})$ in the reduction of x to y . It is this small change which removes all ill-behaviors mentioned above. In addition, the S2a-reducibility is also robust in the sense that several different approaches lead to the same reducibility. The second one (the S1a-reducibility) is also a tolerance-included variation of another reduction S1-reducibility. Both of them are defined on all reals. We will show that the S1a-reducibility coincide with the S2a-reducibility

on the c.a. reals and behave reasonably on the c.a. reals, while S1-reducibility has some ill-behaviors already on the class **WC**. Thus, the S1a-reducibility is a natural extension of the S-reducibility to all reals such that its restrictions on **LC** and **CA** are just the S2a-reducibility and the S-reducibility, respectively. For this reason, we can call the S1a-reducibility (*generalized*) *Solovay reducibility*.

Finally, we show that all reducibilities introduced in this paper imply the rH-reducibility hence have the Solovay property. However, as we show at last, even the rH-reducibility has also a shortcoming that there is no maximal element under the rH-reducibility among the c.a. reals.

2 Straightforward Extension to CA Reals

In this section we investigate the straightforward extension of the S-reducibility to **CA** and analyze why it is not a suitable reducibility to study relative randomness. This extension is obtained directly by replacing the increasing computable sequences by a general computable sequences of rationals in Definition 1.

Definition 2. Let x, y be c.a. reals. x is *S2-reducible* to y ($x \leq_S^2 y$) if there are $(x_s) \in \mathcal{CS}(x)$, $(y_s) \in \mathcal{CS}(y)$ and a constant c such that

$$(\forall s \in \mathbb{N})(|x - x_s| \leq c|y - y_s|). \quad (3)$$

Obviously, \leq_S^2 is reflexive. Actually, it is also transitive on **MC**.

Theorem 1. *S2-reducibility is transitive on the class MC.*

Proof. The result follows from the following claim whose proof is omitted here.

Claim. If $x \leq_S^2 y$ and $y \in \mathbf{MC}$, then for any computable sequence (y_s) which converges to y c_1 -monotonically for some constant c_1 , there are a computable sequences (x_s) and constants c which satisfy condition (3).

Suppose now that $x \leq_S^2 y$, $y \leq_S^2 z$ and $z \in \mathbf{MC}$. Let (z_s) be a computable sequence of rationals which converges to z c_1 -monotonically for some constant $c_1 > 1$. Then, by the above claim, there are $(u_s) \in \mathcal{CS}(y)$ and a constant c_2 which witness the reduction $y \leq_S^2 z$. Suppose furthermore that $(x_s) \in \mathcal{CS}(x)$ and $(y_s) \in \mathcal{CS}(y)$ witness the reduction $x \leq_S^2 y$ with a constant c_3 . Assume w.l.o.g. that z is irrational and any two elements of (z_s) are different. Then define a computable function g by $g(s) := \min\{t > s : |u_t - y_t| \leq |z_s - z_t|\}$. This implies that $|x - x_{g(s)}| \leq c_3|y - y_{g(s)}| \leq c_3(|y - u_{g(s)}| + |u_{g(s)} - y_{g(s)}|) \leq c_3(c_2|z - z_{g(s)}| + |z_s - z_{g(s)}|) \leq c_3(c_2 + 1)/(c_1 - 1)|z - z_s|$. That is, $x \leq_S^2 z$.

Thus, the S2-reducibility is reflexive and transitive on **MC** and it induces a natural S2-degree structure. By the next result, the computable reals form the least S2-degree in this structure.

Theorem 2. *If $x \in \mathbf{EC}$ and $y \in \omega\text{-MC}$, then $x \leq_S^2 y$.*

Moreover, on c.e. reals, the S2-reducibility coincides with the S-reducibility.

Theorem 3. *For any c.e. reals x and y , we have $x \leq_S y$ iff $x \leq_S^2 y$.*

Therefore, S2-reducibility is really an extension of the S-reducibility to **MC**. However, the next theorem shows that the S2-reducibility behaves not satisfactorily outside the class **MC**.

Theorem 4. 1. $(\forall x \in \mathbf{EC} \setminus \mathbb{Q})(\exists y \in \mathbf{WC})(x \not\leq_S^2 y)$;
2. S2-reducibility is not transitive on **WC**.

Proof. 1. For any irrational computable real x , choose a computable sequence (x_s) of rationals such that $|x - x_s| < 2^{-s}$ for all s . We will determine a $y \in \mathbf{WC}$ such that $x \not\leq_S^2 y$. That is, y satisfies all the following requirements

$$R_{\langle i, j, k \rangle} : \left. \begin{array}{l} \varphi_i, \varphi_j \text{ are total} \\ \lim_{s \rightarrow \infty} \varphi_i(s) = x \\ \lim_{s \rightarrow \infty} \varphi_j(s) = y \end{array} \right\} \implies (\exists s)(|x - \varphi_i(s)| > k \cdot |y - \varphi_j(s)|)$$

where (φ_i) is an effective enumeration of all computable functions $\varphi_i : \mathbb{N} \rightarrow \mathbb{Q}$.

To satisfy a single requirement R_e for $e = \langle i, j, k \rangle$, we will define an open witness interval I such that every $y \in I$ satisfies R_e . At the beginning, let $I := (a; b)$ for some $a, b \in \mathbb{Q}$. If the sequence $(\varphi_j(s))$ does not enter I , then we are done because $\lim_{s \rightarrow \infty} \varphi_j(s) \neq y$. Otherwise, suppose that $\varphi_j(t) \in I$ for some t . Since φ_i is total and x is irrational, we have $\varphi_i(t) \neq x = \lim_{s \rightarrow \infty} x_s$. Choose an s large enough such that $|x_s - \varphi_i(t)| \geq 2^{-s+1}$. This implies that $|\varphi_i(t) - x| \geq |\varphi_i(t) - x_s| - |x - x_s| > 2^{-s}$. Let $\epsilon := \min\{2^{-s}/(k+1), \varphi_j(t) - a, b - \varphi_j(t)\}$ and define $J := (\varphi_j(t) - \epsilon; \varphi_j(t) + \epsilon)$. Thus, J is a witness interval of R_e .

To satisfy all requirements simultaneously, we construct a sequence (I_e) of witness intervals by a standard finite injury construction such that I_e is a witness interval, $I_{e+1} \subseteq I_e$ and $\lim_{e \rightarrow \infty} l(I_e) = 0$ where $l(I)$ is the length of the interval I . In this case, the unique common real y of all intervals satisfies all requirements. Furthermore, in order to guarantee that y is weakly computable, the length $l(I_e)$ of I_e should be less than 2^{-2^e} .

2. By item 1, let $x \in \mathbf{EC}$ and $z \in \mathbf{WC}$ such that $x \not\leq_S^2 z$. Let y be a rational number. Then we have obviously that $x \leq_S^2 y$ and $y \leq_S^2 z$.

The shortcomings of the S2-reducibility in Theorem 4 is fatal for a reducibility to classify relative randomness, because it is obvious that computable reals have the least, i.e., no randomness. Therefore, we have to look for other reducibilities if we hope to extend the S-reducibility to a more general class than **MC**.

Remark: The S-reducibility can be characterized by several different ways. For example, $x \leq_S y$ iff for all $(y_s) \in \mathbf{ICS}(y)$, there exists an $(x_s) \in \mathbf{ICS}(x)$, a constant c and a computable function f such that $x - x_s \leq c(y - y_{f(s)})$ for all s ; and iff for all $(x_s) \in \mathbf{ICS}(x)$ and $(y_s) \in \mathbf{ICS}(y)$, there exists a computable function f and a constant c such that $x - x_{f(s)} \leq c(y - y_{f(s)})$ for all s , etc. The corresponding extensions of \leq_S to **CA** can also be based on these characterizations. However, they are all stronger strictly than S2-reducibility and behave even worse. Therefore we do not discuss them here in detail.

3 The Reducibility on CA Reals with Tolerance

The S2-reducibility introduced in the last section does not behave very well outside the class **MC**. A better one will be introduced in this section by introducing a tolerance in the definition of S2-reducibility. Notice that the S-reducibility can be defined equivalently as follows. For c.e. reals x and y , $x \leq_S y$ iff there are a constant c and $(x_s) \in \mathcal{ICS}(x)$ and $(y_s) \in \mathcal{ICS}(y)$ such that $(\forall s)(x - x_s \leq c(y - y_s + 2^{-s}))$. This leads to our following definition.

Definition 3. For $x, y \in \mathbf{CA}$, x is S2a-reducible to y (denoted by $x \leq_S^{2a} y$) if there are $(x_s) \in \mathcal{CS}(x)$, $(y_s) \in \mathcal{CS}(y)$ and a constant c such that

$$(\forall s)(|x - x_s| \leq c(|y - y_s| + 2^{-s})). \quad (4)$$

Theorem 5. 1. The S2a-reducibility is reflexive and transitive on **CA**;

2. $(\forall x, y \in \mathbf{LC})(x \leq_S^{2a} y \iff x \leq_S^2 y \iff x \leq_S y)$;

3. $(\forall x, y)(x \in \mathbf{EC} \ \& \ y \in \mathbf{CA} \implies x \leq_S^{2a} y)$.

Proof. 1. Reflexivity is straightforward. We show now the transitivity. Suppose that, with a common constant c , the computable sequences (x_s) and (y_s) witness the reduction $x \leq_S^{2a} y$ and the computable sequences (u_s) and (z_s) witness the reduction $y \leq_S^{2a} z$. Since both computable sequences (y_s) and (u_s) converge to y , we can define an increasing total computable function $f : \mathbb{N} \rightarrow \mathbb{N}$ such that $|u_{f(s)} - y_{f(s)}| \leq 2^{-s}$ and $s < f(s)$ for all s . Thus, we have

$$\begin{aligned} |x - x_{f(s)}| &\leq c(|y - y_{f(s)}| + 2^{-f(s)}) \leq c(|y - u_{f(s)}| + |u_{f(s)} - y_{f(s)}| + 2^{-s}) \\ &\leq c\left(c(|z - z_{f(s)}| + 2^{-f(s)}) + 2 \cdot 2^{-s}\right) \leq c'(|z - z_{f(s)}| + 2^{-s}) \end{aligned}$$

for $c' = \max\{c^2, 2c\}$. That is, $x \leq_S^{2a} z$.

2. The implications $\leq_S \implies \leq_S^2$ and $\leq_S^2 \implies \leq_S^{2a}$ are straightforward. We prove now the implication $\leq_S^{2a} \implies \leq_S$.

Let x, y be non-computable c.e. reals such that $x \leq_S^{2a} y$. There are a constant c and computable sequences (x_s) and (y_s) of rationals which satisfy (4). The sequences (x_s) and (y_s) are of course not necessarily monotone. However, because x is c.e. but not computable, there is an increasing computable function f such that $(x_{f(s)})$ is an increasing computable sequence which converges to x . Correspondingly, the computable sequence $(y_{f(s)})$ converges to a non-computable c.e. real y and hence there is a computable function g such that $(y_{gf(s)})$ is an increasing computable sequence which converges to y . Obviously, the sequences $(x_{gf(s)})$ and $(y_{gf(s)})$ witness the reduction $x \leq_S^{2a} y$ too. That is, we have

$$x - x_{gf(s)} \leq c(y - y_{gf(s)} + 2^{-gf(s)}) = c(y - (y_{gf(s)} - 2^{-gf(s)})).$$

Since $\lim_{s \rightarrow \infty} (y_{gf(s)} - 2^{-gf(s)}) = y$ and $y_{gf(s)} - 2^{-gf(s)} < y$, there is an increasing computable function h such that (v_s) is an increasing computable sequence converging to y where $v_s := y_{hgf(s)} - 2^{-hgf(s)}$. Accordingly, (u_s) is an increasing

computable sequence converging to x for $u_s := x_{hgf(s)}$. Thus, the sequences (u_s) and (v_s) witness the reduction $x \leq_S y$.

3. is straightforward.

Theorem 5 shows that the S2a-reducibility is a natural extension of the S-reducibility to **CA**. Besides, this reducibility is also robust under slight variations in the definition. As an example, we show one of such equivalent variation.

Theorem 6. *For c.a. reals x and y , $x \leq_S^{2a} y$ if and only if for any $(x_s) \in \mathcal{CS}(x)$ and $(y_s) \in \mathcal{CS}(y)$, there are an increasing computable functions f and a constant c such that $(\forall s)(|x - x_{f(s)}| \leq c(|y - y_{f(s)}| + 2^{-s}))$.*

Proof. The “if” part is easy. We prove now the “only if” part.

Suppose that $x \leq_S^{2a} y$ and this reduction is witnessed by $(u_s) \in \mathcal{CS}(x)$ and $(v_s) \in \mathcal{CS}(y)$ together with a constant $c \geq 1$. For any $(x_s) \in \mathcal{CS}(x)$ and $(y_s) \in \mathcal{CS}(y)$ we define an increasing computable functions f inductively by $f(-1) = 0$ and $f(n+1) := \min\{t > f(n) : |u_t - x_t| \leq 2^{-(n+3)} \text{ \& \& } |y_t - v_t| \leq 2^{-(n+3)}\}$ for all n . The function f is well defined because $\lim_{s \rightarrow \infty} u_s = \lim_{s \rightarrow \infty} x_s = x$ and $\lim_{s \rightarrow \infty} v_s = \lim_{s \rightarrow \infty} y_s = y$. Since $f(s) \geq s+1$, for all s , we have

$$\begin{aligned} |x - x_{f(s)}| &\leq |x - u_{f(s)}| + |u_{f(s)} - x_{f(s)}| \leq c(|y - v_{f(s)}| + 2^{-f(s)}) + 2^{-(s+2)} \\ &\leq c(|y - y_{f(s)}| + |y_{f(s)} - v_{f(s)}| + 2^{-(s+1)}) + 2^{-(s+2)} \\ &\leq c(|y - y_{f(s)}| + 2^{-(s+2)} + 2^{-(s+1)}) + 2^{-(s+2)} \\ &\leq c(|y - y_{f(s)}| + 2^{-s}). \end{aligned}$$

Theorem 7. *If y is random and $x \leq_S^{2a} y$, then $x \leq_S^2 y$.*

Proof. Suppose that $x \leq_S^{2a} y$ and the computable sequences (x_s) and (y_s) witness the reduction with a constant c . Since y is random, there is a constant c_1 such that $|y - y_s| \geq c_1 \cdot 2^{-s}$ for all s . This implies that $|x - x_s| \leq c(|y - y_s| + 2^{-s}) \leq c(1 + c_1)|y - y_s|$ for all s . That is, $x \leq_S^2 y$.

4 Reducibilities on All Reals

In this section we extend the S2a-reducibility to the S1a-reducibility which works on all reals. We will see that also in this case the S1a-reducibility which involves the tolerance works more reasonably than the S1-reducibility (the variation without tolerance). Furthermore, the restrictions of the S1a-reducibility to the classes **CA** and **LC** are just the S2a-reducibility and the S-reducibility, respectively. In addition, S1a-reducibility implies rH-reducibility. At last we show that, even under rH-reducibility, there is no greatest degree of the c.a. reals.

Definition 4. Let x, y be reals. x is *S1-reducible* to y (denoted by $x \leq_S^1 y$) if there exist a partial computable function $f : \subseteq \mathbb{Q} \rightarrow \mathbb{Q}$ and a constant c such that y is a limit point of $\text{dom}(f)$ which satisfies

$$(\forall r \in \text{dom}(f)) (|x - f(r)| \leq c|y - r|). \quad (5)$$

Correspondingly, x is *S1a-reducible* to y ($x \leq_S^{1a} y$), if (5) is replaced by

$$(\forall r \in \text{dom}(f)) \left(|x - f(r)| \leq c(|y - r| + 2^{-l(r)}) \right) \quad (6)$$

where $l(r)$ is the length of the rational r (as a binary word).

Theorem 8. 1. $(\forall x, y \in \mathbb{R})(x \leq_S^2 y \implies x \leq_S^1 y \ \& \ x \leq_S^{2a} y \implies x \leq_S^{1a} y)$;
 2. $(\forall x, y \in \mathbf{CA})(x \leq_S^1 y \iff x \leq_S^2 y \ \& \ x \leq_S^{1a} y \iff x \leq_S^{2a} y)$.

From Theorem 8, the reducibility \leq_S^1 (\leq_S^{1a}) coincides with \leq_S^2 (\leq_S^{2a}) on the class \mathbf{CA} . In addition, the class \mathbf{CA} is closed downward under S1- and S1a-reducibilities. By a finite injury priority construction similar to the proof of Theorem 4.1 we can show the following.

Theorem 9. $(\forall x \in \mathbf{EC} \setminus \mathbb{Q})(\exists y \in \mathbf{WC})(x \not\leq_S^1 y)$

Thus, the reducibility \leq_S^1 is not suitable at least on the class \mathbf{WC} . The S1a-reducibility, on the other hand, behave much better.

Theorem 10. 1. $(\forall x, y \in \mathbb{R})(x \in \mathbf{EC} \implies x \leq_S^{1a} y)$;
 2. *The S1a-reducibility is reflexive and transitive.*

Proof. 1. Let x be a computable real and (x_s) a computable sequence of rationals such that $|x - x_s| \leq 2^{-s}$ for all s . Define $f(r) := x_s$ for $s = l(r)$. Then f is a computable function such that $|x - f(r)| \leq 2^{-l(r)}$ for any $r \in \mathbb{Q}$ and hence f witnesses the reduction $x \leq_S^{1a} y$ with $c := 1$.

2. We prove the transitivity only. Let f and g be computable functions which witness the reduction $x \leq_S^{1a} y$ and $y \leq_S^{1a} z$ together with a common constant c . We construct a Turing machine M which computes a function $h : \subseteq \mathbb{Q} \rightarrow \mathbb{Q}$ as follows: given any $r \in \text{dom}(g)$, M looks for the first $r_1 \in \text{dom}(f)$ such that $|r_1| > |r|$ and $|r_1 - g(r)| < 2^{-|r|}$. Then M outputs $f(r_1)$.

It is not difficult to see that $\text{dom}(h) = \text{dom}(g)$ and hence x is a limit point of $\text{dom}(h)$. In addition, for any $r \in \text{dom}(h)$, we have $|x - h(r)| = |x - f(r_1)| \leq c(|y - r_1| + 2^{-|r_1|}) \leq c(|y - g(r)| + |g(r) - r_1| + 2^{-|r_1|}) \leq c^2(|z - r| + 3 \cdot 2^{-|r_1|}) \leq c'(|z - r| + 2^{-|r|})$. That is, $x \leq_S^{1a} z$.

At last, we compare the S1a-reducibility with the rH-reducibility introduced by Downey, Hirschfeldt and LaForte [6]. Let's recall the definition first.

Definition 5 (Downey, Hirschfeldt and LaForte [6]). For reals x and y , x is rH-reducible to y (denoted $x \leq_{rH} y$) if there are a constant k and a computable function f such that

$$(\forall n \in \mathbb{N})(\exists j \leq k) (f(y \upharpoonright n, j) \downarrow = y \upharpoonright n). \quad (7)$$

It is shown in [6] that the S-reducibility is strictly stronger than the rH-reducibility and the rH-reducibility has the Solovay property. Now we show that the S1a-reducibility implies the rH-reducibility.

Theorem 11. $(\forall x, y \in \mathbb{R})(x \leq_S^{1a} y \implies x \leq_{rH} y)$.

Proof. Suppose that $x \leq_S^{1a} y$ and, w.l.o.g. y is irrational. Let c be a constant and $f : \subseteq \mathbb{Q} \rightarrow \mathbb{Q}$ be a computable function which witness the reduction $x \leq_S^{1a} y$, that is, they satisfy condition (6). For any binary word σ of length n , let

$$S_\sigma := \left\{ \mu \in \{0, 1\}^n : (\exists \tau \in \text{dom}(f)) \left(\sigma \sqsubseteq \tau \ \& \ |f(\tau) - \mu| < 2^{-(n+2)} \right) \right\}.$$

Obviously, S_σ can be computably enumerated uniformly on σ . That is, there is a computable function $\varphi : \subseteq \mathbb{Q} \times \mathbb{N} \rightarrow \mathbb{Q}$ such that $\varphi(\sigma, 0), \varphi(\sigma, 1), \varphi(\sigma, 2), \dots$ are one-one enumerations of S_σ for all σ .

Let N be a natural number such that $c \leq 2^N$ and $\sigma = y \upharpoonright n$ for some n . Now we show that $|S_\sigma| \leq 2^{N+1}$. Suppose that $\tau_1, \tau_2 \in \text{dom}(f)$ are binary words such that $\sigma \sqsubseteq \tau_1, \sigma \sqsubseteq \tau_2$ and $\tau_1 \upharpoonright (n+N+3) = \tau_2 \upharpoonright (n+N+3) = y \upharpoonright (n+N+3)$, then we have $|f(\tau_1) - f(\tau_2)| \leq |f(\tau_1) - x| + |f(\tau_2) - x| \leq c(|\tau_1 - y| + 2^{-l(\tau_1)} + |\tau_2 - y| + 2^{-l(\tau_2)}) \leq 2^N \cdot 4 \cdot 2^{-(n+N+3)} = 2^{-(n+1)}$. Let $u_1, u_2 \in \{0, 1\}^n$ be binary words of S_σ corresponding to τ_1 and τ_2 , respectively. Then we have $|f(\tau_1) - u_1| \leq 2^{-(n+2)}$ and $|f(\tau_2) - u_2| \leq 2^{-(n+2)}$, and hence $|u_1 - u_2| \leq |u_1 - f(\tau_1)| + |u_2 - f(\tau_2)| + |f(\tau_1) - f(\tau_2)| < 2^{-(n+2)} + 2^{-(n+2)} + 2^{-(n+1)} = 2^{-n}$. This implies that $u_1 = u_2$ because the length of u_1 and u_2 are n . On the other hand, there are only 2^{N+2} different binary words of length $n + N + 3$ which extend the word σ of length n . Therefore, $|S_\sigma| \leq 2^{N+2}$ holds.

Finally, since y is a limit point of $\text{dom}(f)$ and irrational, for any n and $\sigma = y \upharpoonright n$, there exists a $\tau \in \text{dom}(f)$ such that $y \upharpoonright (n+N+3) \sqsubseteq \tau$, i.e., $|y - \tau| \leq 2^{-(n+N+3)}$. This implies that $|x - f(\tau)| \leq 2^{-(n+3)}$ and hence $x \upharpoonright n = f(\tau) \upharpoonright n$. Let $u := f(\tau) \upharpoonright n$. Then $|f(\tau) - u| \leq |f(\tau) - x| + |x - u| \leq 2^{-(n+2)}$. That is, u is an element of S_σ and hence there is a $j \leq 2^{N+1}$ such that $\varphi(y \upharpoonright n, j) = u = x \upharpoonright n$. That is, the computable function φ and constant 2^{N+1} witness the reduction $x \leq_{rH} y$.

Corollary 1. *Reducibilities $\leq_S^2, \leq_S^{2a}, \leq_S^1$ and \leq_S^{1a} have the Solovay property.*

In summary, the S1a-reducibility seems to be a quite natural reduction to study relative randomness. Its restrictions on the classes **CA** and **LC** are just the S2a-reducibility and the Solovay reducibility, respectively. The rH-reducibility is a more general and weaker reduction than S1a-reducibility. The next theorem shows that even the rH-reducibility does not have a greatest degree of c.a. reals and hence the c.a. randomness cannot be captured by rH-completeness.

Theorem 12. $(\forall y \in \mathbf{CA})(\exists x \in \mathbf{CA})(x \not\leq_{rH} y)$.

Proof. Let y be a c.a. real number and (y_s) a computable sequence of rationals which converges to y . We construct a computable sequence (x_s) of rationals which converges to x such that $x \not\leq_{rH} y$. That is, they satisfy the following requirements

$$R_{\langle i, j \rangle} \quad : \quad (\exists n \in \mathbb{N})(\forall k \leq j) (\varphi_i(y \upharpoonright n, k) \neq x \upharpoonright n),$$

where (φ_i) is an effective enumeration of all computable functions $\varphi_i : \subseteq \{0, 1\}^* \times \mathbb{N} \rightarrow \{0, 1\}^*$.

To satisfy all requirements simultaneously, we define a sequence (n_e) by $n_{-1} := 0$ and $n_{e+1} := n_e + \pi_2(e) + 2$ for all e first, where π_2 is the second inverse function of the pairing function $\langle \cdot, \cdot \rangle$. That is, $\pi_e(\langle i, j \rangle) = j$. At any stage s , if $\varphi_{i,s}(y_s \upharpoonright n_e, k)$ is defined for some $k \leq j$, then we define a binary words x_s in such a way that

1. $x_{s-1} \upharpoonright n_{e-1} = x_s \upharpoonright n_{e-1}$; and
2. $\varphi_{i,s}(y_s \upharpoonright n_e, k)[n_{e-1} + k] \neq x_s[n_{e-1} + k]$.

Thus, (x_s) is a computable sequence of rationals which converges to a real number x which satisfies all requirements R_e . That is, $x \not\leq_{RH} y$.

References

1. K. Ambos-Spies, K. Weihrauch, and X. Zheng. Weakly computable real numbers. *Journal of Complexity*, 16(4):676–690, 2000.
2. C. S. Calude and P. H. Hertling. Computable approximations of reals: an information-theoretic analysis. *Fund. Inform.*, 33(2):105–120, 1998.
3. C. S. Calude, P. H. Hertling, B. Khoussainov, and Y. Wang. Recursively enumerable reals and Chaitin Ω numbers. *Theoretical Computer Science*, 255:125–149, 2001.
4. G. Chaitin. A theory of program size formally identical to information theory. *J. of ACM.*, 22:329–340, 1975.
5. R. G. Downey and D. R. Hirschfeldt. *Algorithmic Randomness and Complexity*. Springer-Verlag, 200? monograph to be published.
6. R. G. Downey, D. R. Hirschfeldt, and G. LaForte. Randomness and reducibility. In J. Sgall, A. Pultr, and P. Kolman, editors, *MFCS 2001, Mariánské Lázně, Czech Republic, August 27-31, 2001*, volume 2136 of *LNCS*, pages 316–327. Springer, 2001.
7. R. G. Downey, D. R. Hirschfeldt, and A. Nies. Randomness, computability, and density. *SIAM J. Comput.*, 31(4):1169–1183 (electronic), 2002.
8. A. Kučera and T. A. Slaman. Randomness and recursive enumerability. *SIAM J. Comput.*, 31(1):199–211, 2001.
9. P. Martin-Löf. The definition of random sequences. *Information and Control*, 9:602–619, 1966.
10. R. Rettinger and X. Zheng. On the hierarchy and extension of monotonically computable real numbers. *J. Complexity*, 19(5):672–691, 2003.
11. R. Rettinger, X. Zheng, R. Gengler, and B. von Braunmühl. Monotonically computable real numbers. *Math. Log. Quart.*, 48(3):459–479, 2002.
12. R. M. Solovay. Draft of a paper (or a series of papers) on chaitin's work manuscript, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, p. 215, 1975.
13. A. M. Turing. On computable numbers, with an application to the “Entscheidungsproblem”. *Proceedings of the London Mathematical Society*, 42(2):230–265, 1936.
14. L. Yu and D. Ding. There is no sw-complete c.e. real. *J. of Symbolic Logic*. (to appear).

The Complexity of Counting Solutions to Systems of Equations over Finite Semigroups

Gustav Nordh* and Peter Jonsson**

Department of Computer and Information Science
Linköpings Universitet
S-581 83 Linköping, Sweden
Fax: +46 13 28 44 99
{gusno,petej}@ida.liu.se

Abstract. We study the computational complexity of counting the number of solutions to systems of equations over a fixed finite semigroup. We show that if the semigroup is a group, the problem is tractable if the group is Abelian and $\#P$ -complete otherwise. If the semigroup is a monoid (that is not a group) the problem is $\#P$ -complete. In the case of semigroups where all elements have divisors we show that the problem is tractable if the semigroup is a direct product of an Abelian group and a rectangular band, and $\#P$ -complete otherwise. The class of semigroups where all elements have divisors contains most of the interesting semigroups e.g. regular semigroups. These results are proved by the use of powerful techniques from universal algebra.

1 Introduction

The computational complexity of deciding whether systems of equations over a fixed finite semigroup are solvable has been intensively studied in the past. In [3], it is proved that when the semigroup is a group, the problem is tractable if the group is Abelian and NP-complete otherwise. This line of research continued in [6, 7], where the corresponding problem for finite monoids was given a complete solution. Some partial results in the general case of finite semigroups have been proved in [7]. Note that even the restricted problem of determining the computational complexity of solving systems of equations over a fixed regular semigroup is still open.

In this paper we study the computational complexity of counting solutions to systems of equations over a fixed finite semigroup. This problem has not been given the same attention as the corresponding decision problem. A system of equations over a fixed finite semigroup (S, \cdot) is a collection of equations of the form $w_1 \cdot w_2 \cdot \dots \cdot w_k = y_1 \cdot y_2 \cdot \dots \cdot y_j$ where each w_i and y_i is either a variable or a constant in S . We denote the problem of counting the number of solutions

* Supported by the *National Graduate School in Computer Science* (CUGS), Sweden.

** Partially supported by the Swedish Research Council (VR) under grant 221-2000-361.

to systems of equations over the semigroup (S, \cdot) by $\#EQN_S^*$. The notation $\#EQN_S^*$ is used in order to comply with the notation in [3, 6, 7]. The presence of the $*$ in $\#EQN_S^*$ is motivated by the need to distinguish this problem from the problem of counting the number of solutions to a single equation over (S, \cdot) (denoted by $\#EQN_S$). The complexity of $\#EQN_S^*$ is measured in the size of the systems of equations (the size of (S, \cdot) is fixed and does not matter).

We prove that $\#EQN_G^*$ is tractable when (G, \cdot) is an Abelian group and $\#P$ -complete when (G, \cdot) is a non-Abelian group. In the case of monoids, $\#EQN_M^*$ is $\#P$ -complete when (M, \cdot) is a monoid that is not a group. Given a semigroup (S, \cdot) , we say that an element a in S is divisible if and only if there exists b, c in S such that $b \cdot c = a$. In the case of semigroups where all elements are divisible we show that $\#EQN_S^*$ is tractable if (S, \cdot) is a direct product of an Abelian group and a rectangular band, and $\#P$ -complete otherwise. The class of semigroups where all elements have divisors contains most of the interesting semigroups, in particular regular semigroups. Moreover we prove that if (S, \cdot) is a semigroup with zero, then $\#EQN_S^*$ is tractable if (S, \cdot) is such that for all elements x, y in S , $x \cdot y = 0$, and $\#P$ -complete otherwise. Important to note is that all the conditions we impose on our semigroups to ensure the tractability (or $\#P$ -completeness) of $\#EQN_S^*$ can be checked in polynomial time (in the size of the semigroups).

Our approach for obtaining dichotomy theorems for $\#EQN_S^*$ relies heavily on the use of powerful algebraic techniques that have previously shown to be very useful in proving dichotomy theorems for Constraint Satisfaction Problems (CSPs) [1, 2]. It is our belief that similar algebraic techniques can also be used in a unified approach to determine the complexity of the decision problem.

The paper is organized as follows. In Section 2 we give the basic definitions and results needed in the rest of this paper. Section 3 contains all our tractability proofs for $\#EQN_S^*$. In Section 4 we prove the dichotomy for the computational complexity of $\#EQN_S^*$ when (S, \cdot) is a group or monoid. In Section 5 we prove the dichotomy for the computational complexity of $\#EQN_S^*$ when (S, \cdot) is a semigroup with zero or is in the class of semigroups where all elements are divisible.

2 Preliminaries

2.1 CSPs

In this section we introduce the notation and basic results on CSPs that we will use in the rest of this paper.

Let D be a finite set. The set of all n -tuples of elements of D is denoted by D^n . Any subset of D^n is called an n -ary relation on D . The set of all finitary relations over D is denoted by R_D .

Definition 1. A constraint language over a finite set, D , is an arbitrary set $\Gamma \subseteq R_D$.

Definition 2. The counting constraint satisfaction problem over the constraint language Γ , denoted $\#CSP(\Gamma)$, is defined to be the counting problem with instance (V, D, C) , where

- V is a finite set of variables,
- D is a set of values (sometimes called a domain), and
- C is a set of constraints $\{C_1, \dots, C_q\}$, in which each constraint C_i is a pair (s_i, ϱ_i) with s_i a list of variables of length m_i , called the constraint scope, and ϱ_i an m_i -ary relation over the set D , belonging to Γ , called the constraint relation.

We ask for the number of solutions to (V, D, C) , that is, the number of functions from V to D such that, for each constraint in C , the image of the constraint scope is a member of the constraint relation.

The corresponding decision problem where we are interested in the existence of a solution is denoted $CSP(\Gamma)$. In the context of this paper D is the elements of a finite semigroup (S, \cdot) and Γ is the set of relations expressible as equations over (S, \cdot) (denoted by Γ_S). For example the equation $x \cdot y = c$ where x, y are variables and c is a constant gives rise to the following relation $R = \{(x, y) | x \cdot y = c\}$.

Next we consider operations on the domain D . Any operation on D can be extended in a standard way to an operation on tuples over D , as follows.

Definition 3. Let f be a k -ary operation on D and let R be an n -ary relation over D . For any collection of k tuples, $t_1, t_2, \dots, t_k \in R$, the n -tuple $f(t_1, t_2, \dots, t_k)$ is defined as follows: $f(t_1, t_2, \dots, t_k) =$

$$(f(t_1[1], t_2[1], \dots, t_k[1]), f(t_1[2], t_2[2], \dots, t_k[2]), \dots, f(t_1[n], t_2[n], \dots, t_k[n]))$$

where $t_j[i]$ is the i :th component in tuple t_j .

A technique that has shown to be useful in determining the computational complexity of $CSP(\Gamma)$ is that of investigating whether Γ is closed under certain families of operations [5].

Definition 4. For any constraint relation $\varrho_i \in \Gamma$ if f is an operation such that for all $t_1, t_2, \dots, t_k \in \varrho_i$: $f(t_1, t_2, \dots, t_k) \in \varrho_i$, then ϱ_i is closed under f . If all constraint relations in Γ are closed under f then Γ is closed under f . An operation f such that Γ is closed under f is called a polymorphism of Γ . The set of all polymorphisms of Γ is denoted $Pol(\Gamma)$.

In this paper we will only be concerned with a particular family of ternary operations called Mal'tsev operations. An operation M is a Mal'tsev operation if and only if for all x, y , $M(x, y, y) = M(y, y, x) = x$.

The following result provides us with the key to prove the #P-completeness of constraint languages.

Theorem 1 ([2]). If Γ is a finite constraint language that is closed under no Mal'tsev operation then $\#CSP(\Gamma)$ is #P-complete.

Throughout this paper we will use the following notation for the application of an arbitrary Mal'tsev operation M on tuples in a relation R .

$$\begin{array}{c} (a, b) \in R \\ (b, b) \in R \\ (b, a) \in R \\ \hline M \\ (a, a) \end{array}$$

In the example above (a, b) , (b, b) and (b, a) are three tuples in a relation R . If $(a, a) \notin R$ we have shown that R is closed under no Mal'tsev operation, since $(M(a, b, b), M(b, b, a)) = (a, a)$ for *any* Mal'tsev operation M .

All of our #P-completeness proofs for $\#EQN_S^*$ follows this outline: First we choose a relation R in Γ_S i.e., R is expressed by a specific equation E over (S, \cdot) . Then we show that R is closed under no Mal'tsev operation, thus proving the #P-completeness of $\#CSP(\Gamma_S)$. It should be clear that this implies the #P-completeness of $\#EQN_S^*$. It is also easy to realize that any system of equations over a finite semigroup (S, \cdot) can be (parsimoniously) reduced in polynomial time to a system of equations over (S, \cdot) such that no equation in this system of equations contains more than three variables. Equations containing more than three variables can be split into smaller parts by introducing new variables. Hence we can safely assume that all constraint languages we consider are finite.

The following theorem is very useful when it comes to proving tractability results for $\#CSP(\Gamma)$.

Theorem 2 ([2]). *A Mal'tsev operation $M(x, y, z) = x * y^{-1} * z$ where $*$ and $^{-1}$ are the operations of an Abelian group is called an affine operation. If $M \in \text{Pol}(\Gamma)$ then $\#CSP(\Gamma)$ is tractable.*

2.2 Definitions and Results from Semigroup Theory

For the convenience of the reader we here give the definitions and results from semigroup theory that we use in the rest of this paper. The proofs are available in (or easily deducible from) any standard reference on semigroup theory such as [4].

A *semigroup* (S, \cdot) is a set S with an associative binary operation \cdot . A *semigroup with zero* is a semigroup that contains an element 0 such that for all x in S , $0 \cdot x = x \cdot 0 = 0$. A *monoid* is a semigroup that has an identity element. A *band* is a semigroup where all elements are idempotent (that is for all x in S , $x \cdot x = x$). A *rectangular band* is a band where for all a, b in S , $a \cdot b \cdot a = a$. A *regular semigroup* is a semigroup where for all a in S there exists an element b in S such that $a \cdot b \cdot a = a$. A *right zero semigroup* is a semigroup where for all a, b in S , $a \cdot b = b$ (*left zero semigroups* are defined in an analogous way). If two semigroups (S, \cdot_s) and (T, \cdot_t) are *isomorphic* we denote this by $(S, \cdot_s) \cong (T, \cdot_t)$. If (S, \cdot_s) and (T, \cdot_t) are semigroups, then the Cartesian product $(S, \cdot_s) \times (T, \cdot_t)$ becomes a semigroup if we define $(s, t) \cdot (s', t') = (s \cdot_s s', t \cdot_t t')$. We refer to this semigroup as the *direct product* of (S, \cdot_s) and (T, \cdot_t) . Let $\text{Div}(S)$ denote the set

of divisible elements in S where (S, \cdot) is a finite semigroup (a is divisible if and only if there exists b, c in S such that $b \cdot c = a$).

Theorem 3. *If (S, \cdot) is a finite semigroup and a an arbitrary element in S , then a^m is idempotent for some m .*

Theorem 4. *If (M, \cdot) is a finite monoid with the identity element as the only idempotent then (M, \cdot) is a group.*

Theorem 5. *Let (S, \cdot) be a rectangular band, then there exist a left zero semigroup (L, \cdot_l) and a right zero semigroup (R, \cdot_r) such that $(S, \cdot) \cong (L, \cdot_l) \times (R, \cdot_r)$.*

The following theorem is folklore.

Theorem 6. *A regular semigroup (S, \cdot) whose idempotents form a rectangular band is isomorphic to a direct product of a group and a rectangular band.*

We give some hints on how to prove this theorem. By using Green's relations it can be proved that (S, \cdot) is a completely simple semigroup, and since (S, \cdot) is orthodox it follows from the Rees-Suschkewitch Theorem [4] that (S, \cdot) is a direct product of a group and the rectangular band of its idempotents.

3 Tractability Results

Due to the similar structure of the tractability proofs for the tractable cases of $\#EQN_S^*$ we have gathered all of these proofs in this section to make a unified approach possible. In the rest of this section assume that all equations in the systems of equations are of the following two types: equations only containing variables, $x_1 \cdot \dots \cdot x_n = \hat{x}_1 \cdot \dots \cdot \hat{x}_m$ or equations with a single variable on the left hand side and a constant on the right hand side, $x = a$. We can make this assumption because it is easy to reduce (parsimoniously and in polynomial time) any system of equations over a finite semigroup into a system of equations of this form.

Let A be finite set and let $f : A^n \rightarrow A$ and $g : A^m \rightarrow A$ be two operations. If $f(a, \dots, a) = a$ for all $a \in A$, then f is *idempotent*. We say that g *commutes* with f if for all $a_{11}, \dots, a_{1m}, \dots, a_{nm} \in A$,

$$g(f(a_{11}, \dots, a_{n1}), \dots, f(a_{1m}, \dots, a_{nm})) = f(g(a_{11}, \dots, a_{1m}), \dots, g(a_{n1}, \dots, a_{nm}))$$

The following lemma is the foundation for our tractability results.

Lemma 1. *Let (S, \cdot) be a finite semigroup. If $f : S^k \rightarrow S$ is an idempotent operation and \cdot commutes with f , then $f \in \text{Pol}(\Gamma_S)$.*

Proof. Arbitrarily choose a relation $R \in \Gamma_S$. If R is a unary relation (i.e. $R(x)$ holds if and only if $x = a$ for some $a \in S$), then R is closed under f since f is idempotent. Otherwise, $R = \{(x_1, \dots, x_n, \hat{x}_1, \dots, \hat{x}_m) \mid x_1 \cdot \dots \cdot x_n = \hat{x}_1 \cdot \dots \cdot \hat{x}_m\}$

and we arbitrarily choose k tuples t_1, \dots, t_k in R . Now, R is closed under f if the following holds:

$$\prod_{i=1}^n f(t_1[i], \dots, t_k[i]) = \prod_{i=n+1}^{n+m} f(t_1[i], \dots, t_k[i]) \quad (*)$$

We know that \cdot commutes with f so

$$\prod_{i=1}^n f(t_1[i], \dots, t_k[i]) = f\left(\prod_{i=1}^n t_1[i], \dots, \prod_{i=1}^n t_k[i]\right)$$

and

$$\prod_{i=n+1}^{n+m} f(t_1[i], \dots, t_k[i]) = f\left(\prod_{i=n+1}^{n+m} t_1[i], \dots, \prod_{i=n+1}^{n+m} t_k[i]\right).$$

Since t_1, \dots, t_k are tuples in R , it follows that $\prod_{i=1}^n t_j[i] = \prod_{i=n+1}^{n+m} t_j[i]$ for all j and $(*)$ holds. \square

Lemma 2. *Let (S, \cdot) be a finite semigroup. If f is an affine operation and \cdot commutes with f , then $\#EQN_S^*$ is tractable.*

Proof. We see that f is idempotent since $f(a, a, a) = a * a^{-1} * a = a$ so $f \in \text{Pol}(\Gamma_S)$ by Lemma 1. Consequently, Theorem 2 implies that $\#EQN_S^*$ is tractable. \square

Our tractability results for $\#EQN_S^*$ are contained in the following theorem.

Theorem 7. *$\#EQN_S^*$ is tractable when*

1. (S, \cdot) is an Abelian group.
2. (S, \cdot) is a left zero or right zero semigroup.
3. (S, \cdot) is a semigroup with zero such that for all x, y in S , $x \cdot y = 0$.
4. $(S, \cdot) \cong (T, *) \times (T', *')$ and $\#EQN_T^*$, $\#EQN_{T'}^*$ are tractable.

Proof. If (S, \cdot) is an Abelian group, then let $M(x, y, z) = x \cdot y^{-1} \cdot z$. We show that \cdot commutes with M and tractability follows from Lemma 2. Let a, b, c be arbitrary tuples in S^2 , then $M(a[1], b[1], c[1]) \cdot M(a[2], b[2], c[2]) = a[1] \cdot b[1]^{-1} \cdot c[1] \cdot a[2] \cdot b[2]^{-1} \cdot c[2] = a[1] \cdot a[2] \cdot (b[1] \cdot b[2])^{-1} \cdot c[1] \cdot c[2] = M(a[1] \cdot a[2], b[1] \cdot b[2], c[1] \cdot c[2])$.

If (S, \cdot) is a right-zero group (the proof for left-zero groups is analogous), then let $M(x, y, z) = x * y^{-1} * z$ where the operation $*$ is chosen such that $(S, *)$ is an Abelian group (such a choice is always possible). Let a, b, c be arbitrary tuples in S^2 , then $M(a[1], b[1], c[1]) \cdot M(a[2], b[2], c[2]) = M(a[2], b[2], c[2]) = M(a[1] \cdot a[2], b[1] \cdot b[2], c[1] \cdot c[2])$ and tractability follows from Lemma 2.

If (S, \cdot) is a semigroup with zero such that for all x, y in S , $x \cdot y = 0$, then let $M(x, y, z) = x * y^{-1} * z$ where the operation $*$ is chosen such that $(S, *)$ is an Abelian group. Let a, b, c be arbitrary tuples in S^2 , then $M(a[1], b[1], c[1]) \cdot$

$M(a[2], b[2], c[2]) = 0 = M(a[1] \cdot a[2], b[1] \cdot b[2], c[1] \cdot c[2])$ and tractability follows from Lemma 2.

If $(S, \cdot) \cong (T, *) \times (T', *')$, then a system of equations over (S, \cdot) can be split into two independent systems of equations, one over $(T, *)$ and one over $(T', *')$. The number of solutions to the system of equations over (S, \cdot) equals the product of the number of solutions to the systems of equations over $(T, *)$ and $(T', *')$ respectively. Hence if $\#EQN_T^*$ and $\#EQN_{T'}^*$ are tractable, then so is $\#EQN_S^*$. \square

4 Finite Groups and Monoids

In this section we prove our dichotomies for the complexity of the counting problem over finite groups and monoids. Note that the decision dichotomy proved in [6] and the counting dichotomy we prove differs in the case of monoids, for example it is tractable to decide whether systems of equations over a commutative band are solvable, but to count the number of solutions is $\#P$ -complete.

Theorem 8. *Let (G, \cdot) be a finite group, then $\#EQN_G^*$ is tractable if (G, \cdot) is Abelian, and $\#P$ -complete otherwise.*

Proof. We begin by proving the $\#P$ -completeness of $\#EQN_G^*$ when (G, \cdot) is a non-Abelian group. Because (G, \cdot) is a non-Abelian group there exists two elements a, b in G such that $a \cdot b \neq b \cdot a$. Consider the following relation $R = \{(x, y) | x \cdot y = y \cdot x\}$. It is easy to see that R is closed under no Mal'tsev operation.

$$\begin{array}{ll} (a, 1) \in R & a \cdot 1 = 1 \cdot a \\ (1, 1) \in R & 1 \cdot 1 = 1 \cdot 1 \\ (1, b) \in R & 1 \cdot b = b \cdot 1 \\ \hline M & \\ (a, b) \notin R & a \cdot b \neq b \cdot a \end{array}$$

Hence $\#EQN_G^*$ is $\#P$ -complete for non-Abelian groups. The tractability part is proved in Theorem 7. \square

Next we consider the case where the semigroup is a finite monoid.

Theorem 9. *Let (M, \cdot) be a finite monoid that is not a group, then $\#EQN_M^*$ is $\#P$ -complete.*

Proof. According to Theorem 4 we know that there exist an element a in M such that $a \neq 1$ and $aa = a$. Consider the relation $R = \{(x, y) | x \cdot y = a\}$. It is easy to see that R is closed under no Mal'tsev operation.

$$\begin{array}{ll} (1, a) \in R & 1 \cdot a = a \\ (a, a) \in R & a \cdot a = a \\ (a, 1) \in R & a \cdot 1 = a \\ \hline M & \\ (1, 1) \notin R & 1 \cdot 1 = 1 \neq a \end{array}$$

Thus, $\#EQN_M^*$ is $\#P$ -complete. \square

5 Finite Semigroups

The computational complexity of deciding whether systems of equations over a fixed finite semigroup are solvable has been studied in [7], where some evidence is given that a dichotomy in the decision case would be very hard to prove. Even the restricted problem of proving a dichotomy for regular semigroups is open. We prove that for semigroups where all elements have divisors the counting problem is tractable if the semigroup is a direct product of an Abelian group and a rectangular band, and $\#P$ -complete otherwise. Moreover we prove that for semigroups with zero, the problem is tractable if the semigroup is such that for all elements x, y $x \cdot y = 0$, and $\#P$ -complete otherwise.

The following lemma will be very useful in the proofs that follows.

Lemma 3. *Let (S, \cdot) be a semigroup and I be the set of idempotents in S . A necessary condition for $\#EQN_S^*$ to be tractable is that the following holds: for all a in I and for all b, c in S , $b \cdot c = b \cdot a \cdot c$. Otherwise $\#EQN_S^*$ is $\#P$ -complete.*

Proof. Assume that there exists a in I and that there exists b, c in S and $b \cdot c \neq b \cdot a \cdot c$. Consider the relation $R = \{(x, y) | x \cdot y = b \cdot a \cdot c\}$. R is closed under no Mal'tsev operation.

$$\begin{array}{ccc} (b, a \cdot c) & \in R & b \cdot a \cdot c = b \cdot a \cdot c \\ (b \cdot a, a \cdot c) & \in R & b \cdot a \cdot a \cdot c = b \cdot a \cdot c \\ (b \cdot a, c) & \in R & b \cdot a \cdot c = b \cdot a \cdot c \\ \hline M & & \\ (b, c) & \notin R & b \cdot c \neq b \cdot a \cdot c \end{array}$$

Hence $\#EQN_S^*$ is $\#P$ -complete. \square

We continue by proving a dichotomy in the case where the semigroup is a band.

Theorem 10. *If (S, \cdot) is a band, then $\#EQN_S^*$ is tractable if (S, \cdot) is a rectangular band and $\#P$ -complete otherwise.*

Proof. It follows directly from Lemma 3 that if S is a band that is not a rectangular band then $\#EQN_S^*$ is $\#P$ -complete. If (S, \cdot) is a rectangular band we know from Theorem 5 that there exist a left zero semigroup (L, \cdot_l) and a right zero semigroup (R, \cdot_r) such that $(S, \cdot) \cong (L, \cdot_l) \times (R, \cdot_r)$. Thus the tractability of $\#EQN_S^*$ when (S, \cdot) is a rectangular band follows from Theorem 7. \square

The following lemma gives us an important necessary condition for the tractability of $\#EQN_S^*$.

Lemma 4. *Given a semigroup (S, \cdot) , it is a necessary condition for $\#EQN_S^*$ to be tractable that $(Div(S), \cdot)$ form a regular semigroup. Otherwise $\#EQN_S^*$ is $\#P$ -complete.*

Proof. Arbitrarily choose $a \in \text{Div}(S)$ and assume that $a = b \cdot c$ for some b, c . Since S is finite, we know from Theorem 3 that b^n and c^m are idempotent for some m, n . From Lemma 3 we know that $a = b \cdot c = b \cdot c^m \cdot b^n \cdot c = a \cdot c^{m-1} \cdot b^{n-1} \cdot a$, otherwise $\#EQN_S^*$ is $\#P$ -complete. Thus it is a necessary condition for $\#EQN_S^*$ to be tractable that all elements in $\text{Div}(S)$ are regular. \square

Now we can prove a dichotomy $\#EQN_S^*$ when (S, \cdot) is a regular semigroup.

Theorem 11. *Let (S, \cdot) be a finite regular semigroup. Then $\#EQN_S^*$ is tractable if (S, \cdot) is a direct product of an Abelian group and a rectangular band. Otherwise $\#EQN_S^*$ is $\#P$ -complete.*

Proof. If the idempotents in (S, \cdot) do not form a rectangular band then by Lemma 3, $\#EQN_S^*$ is $\#P$ -complete. If the idempotents in (S, \cdot) do form a rectangular band then by Theorem 6, (S, \cdot) is a direct product of a group and a rectangular band. We conclude from Theorem 7 that $\#EQN_S^*$ is tractable if (S, \cdot) is a direct product of an Abelian group and a rectangular band.

Now assume that (S, \cdot) is isomorphic to the direct product of a non-Abelian group by a rectangular band i.e. $(S, \cdot) \cong (G, \cdot_g) \times (RB, \cdot_b)$ where (G, \cdot_g) is a non-Abelian group and (RB, \cdot_b) is a rectangular band. Consider the following relation $R = \{(x, y) \mid x \cdot y = y \cdot x\}$. We show that R is closed under no Mal'tsev operation. The elements below are chosen as follows, 1 is the identity in (G, \cdot_g) , since (G, \cdot_g) is non-Abelian we know that there exists elements a, b in G such that $a \cdot_g b \neq b \cdot_g a$, and c is an arbitrary element in (RB, \cdot_b) .

$$\begin{array}{lcl} ((a, c), (1, c)) \in R & (a, c) \cdot (1, c) = (1, c) \cdot (a, c) \\ ((1, c), (1, c)) \in R & (1, c) \cdot (1, c) = (1, c) \cdot (1, c) \\ ((1, c), (b, c)) \in R & (1, c) \cdot (b, c) = (b, c) \cdot (1, c) \\ \hline M & & \\ \hline ((a, c), (b, c)) \notin R & (a, c) \cdot (b, c) \neq (b, c) \cdot (a, c) \end{array}$$

Hence $\#EQN_S^*$ is $\#P$ -complete when (S, \cdot) is isomorphic to the direct product of a non-Abelian group by a rectangular band. \square

The following corollary follows directly from Lemma 4 and Theorem 11.

Corollary 1. *If (S, \cdot) is a finite semigroup where all elements have divisors, then $\#EQN_S^*$ is tractable if (S, \cdot) is a direct product of an Abelian group and a rectangular band, and $\#P$ -complete otherwise.*

The following theorem proves a dichotomy for $\#EQN_S^*$ when (S, \cdot) is a semigroup with zero.

Theorem 12. *Let (S, \cdot) be a finite semigroup with zero, then $\#EQN_S^*$ is tractable if for all x, y in S , $x \cdot y = 0$, and $\#P$ -complete otherwise.*

Proof. The tractability part is proved in Theorem 7. We continue by proving the $\#P$ -completeness of $\#EQN_S^*$ when (S, \cdot) is a semigroup with zero and there exist two elements a, b in S such that $a \cdot b \neq 0$. Consider the following relation

$R = \{(x, y) | x \cdot y = 0\}$. It is easy to see that R is closed under no Mal'tsev operation.

$$\begin{array}{lcl} (a, 0) \in R & a \cdot 0 = 0 \\ (0, 0) \in R & 0 \cdot 0 = 0 \\ (0, b) \in R & 0 \cdot b = 0 \\ \hline M & & \\ (a, b) \notin R & a \cdot b \neq 0 \end{array}$$

Hence $\#EQN_S^*$ is $\#P$ -complete when (S, \cdot) is a semigroup with zero and there exist two elements x, y in S such that $x \cdot y \neq 0$. \square

It is easy to realize that all the conditions we impose on our semigroups to ensure the tractability (or $\#P$ -completeness) of $\#EQN_S^*$ can be checked in polynomial time (in the size of the semigroups), except maybe for the condition in Theorem 11. To check whether a semigroup, isomorphic to a direct product of a group and a rectangular band, is in fact isomorphic to a direct product of an Abelian group and a rectangular band, we need a way to extract the group part from the semigroup. Choose an idempotent x in (S, \cdot) . Then we know that x is of the form $(1, x')$ where 1 is the identity in (G, \cdot_g) . Consider the set xSx , elements in this set is of the form $(1, x') \cdot (a, y) \cdot (1, x') = (a, x')$. This implies that $(xSx, \cdot) \cong (G, \cdot_g)$. Now it is easy to check whether (G, \cdot_g) is Abelian.

Acknowledgments

The authors want to thank Pascal Tesson for interesting discussions and remarks on the topics in this paper, Andrei Bulatov for many valuable remarks on an earlier version of this paper and Jorge Almeida for his explanations of some topics in the realm of semigroups.

References

1. A. Bulatov. A dichotomy theorem for constraints on a three-element set. In *Proc. 43rd IEEE Symposium on Foundations of Computer Science, FOCS'02*, pages 649–658, 2002.
2. A. Bulatov and V. Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. In *Proc. 44th IEEE Symposium on Foundations of Computer Science, FOCS'03*, 2003.
3. M. Goldmann and A. Russel. The complexity of solving equations over finite groups. *Inform. and Comput.*, 178(1):253–262, 2002.
4. J.M. Howie. *Fundamentals of Semigroup Theory*. Clarendon Press, Oxford, 1995.
5. P. Jeavons, D. Cohen, and M. Gyssens. Closure properties of constraints. *Journal of the ACM*, 44:527–548, 1997.
6. C. Moore, P. Tesson, and D. Thérien. Satisfiability of systems of equations over finite monoids. In *Proc. 26th International Symposium on Mathematical Foundations of Computer Science, MFCS'01*, pages 537–547, 2001.
7. P. Tesson. *Computational Complexity Questions Related to Finite Monoids and Semigroups*. PhD thesis, School of Computer Science, McGill University, Montreal, 2003.

Computational Complexity Classification of Partition under Compaction and Retraction

Narayan Vikas

School of Computing Science, Simon Fraser University, Burnaby,
British Columbia, Canada V5A 1S6
vikas@cs.sfu.ca

Abstract. The compaction and retraction problems are special graph colouring problems, and can also be viewed as partition problems with certain properties. A very close relationship between the compaction, retraction, and constraint satisfaction problems has been established earlier providing evidence that it is likely to be difficult to give a complete computational complexity classification of the compaction and retraction problems for reflexive or bipartite graphs. In this paper, we give a complete computational complexity classification of the compaction and retraction problems for all graphs (including partially reflexive graphs) with four or fewer vertices. The complexity classification of both the compaction and retraction problems is found to be the same for each of these graphs. This relates to a long-standing open problem concerning the equivalence of the compaction and retraction problems. The study of the compaction and retraction problems for graphs with at most four vertices has a special interest as it covers a popular open problem in relation to the general open problem. We also give complexity results for some general graphs.

1 Introduction

We first introduce the following definitions and problems, and then describe the motivation and results.

1.1 Definitions

An edge with the same endpoints in a graph is called a *loop*. A vertex v of a graph is said to have a loop if vv is an edge of the graph. A *reflexive graph* is a graph in which every vertex has a loop. An *irreflexive graph* is a graph which has no loops. Any graph, in general, is a *partially reflexive graph*, meaning that its individual vertices may or may not have loops. Thus reflexive and irreflexive graphs are special partially reflexive graphs. A bipartite graph is irreflexive by definition. We denote an irreflexive complete graph with k vertices by K_k . For a graph G , we use $V(G)$ and $E(G)$ to denote its vertex set and edge set respectively. Given an induced subgraph H of a graph G , we denote by $G - H$, the subgraph obtained by deleting from G the vertices of H together with the edges incident with them; thus $G - H$ is a subgraph of G induced by $V(G) - V(H)$. In the following, let G and H be graphs.

A *homomorphism* $f : G \rightarrow H$, of G to H , is a mapping f of the vertices of G to the vertices of H , such that $f(g)$ and $f(g')$ are adjacent vertices of H whenever g and g' are adjacent vertices of G . If there exists a homomorphism of G to H then G is said to be *homomorphic* to H . Note that for any homomorphism $f : G \rightarrow H$, if a vertex v of G has a loop then the vertex $f(v)$ of H necessarily also has a loop. If G is irreflexive then clearly G is k -colourable if and only if G is homomorphic to K_k . Thus the concept of a homomorphism generalises the concept of a k -colourability.

A *compaction* $c : G \rightarrow H$, of G to H , is a homomorphism of G to H , such that for every vertex x of H , there exists a vertex v of G with $c(v) = x$, and for every edge hh' of H , $h \neq h'$, there exists an edge gg' of G with $c(g) = h$ and $c(g') = h'$. Notice that the first part of the definition for compaction (the requirement for every vertex x of H) is relevant only if H has isolated vertices. If there exists a compaction of G to H then G is said to *compact* to H . Given a compaction $c : G \rightarrow H$, if for a vertex v of G , we have $c(v) = x$, where x is a vertex of H , then we say that the vertex v of G *covers the vertex* x of H under c ; and if for an edge gg' of G , we have $c(\{g, g'\}) = \{h, h'\}$, where hh' is an edge of H , then we say that the edge gg' of G *covers the edge* hh' of H under c (note in the definition of compaction, it is not necessary that a loop of H be covered by any edge of G under c).

We note that the notion of a *homomorphic image* used in [Harary, 1969] (also cf. [Hell and Miller, 1979]) coincides with the notion of a compaction in case of irreflexive graphs (i.e., when G and H are irreflexive in the above definition for compaction).

A *retraction* $r : G \rightarrow H$, of G to H , with H as an induced subgraph of G , is a homomorphism of G to H , such that $r(h) = h$, for every vertex h of H . If there exists a retraction of G to H then G is said to *retract* to H . Note that every retraction $r : G \rightarrow H$ is necessarily also a compaction but not vice versa.

1.2 Homomorphism, Compaction, and Retraction Problems

The problem of deciding the existence of a homomorphism to a fixed graph H , called the *H -colouring problem*, and denoted as *H -COL*, asks whether or not an input graph G is homomorphic to H . If H is a graph with a loop then every graph is trivially homomorphic to H . Also, if H is a bipartite graph then we note that a graph G is homomorphic to H if and only if G is also bipartite and H has an edge if G has an edge. Thus the problem *H -COL* is interesting only if H is an irreflexive non-bipartite graph. A complete complexity classification of *H -COL* is given in [Hell and Nesetril, 1990]. It is shown there that *H -COL* is NP-complete for any irreflexive non-bipartite graph H . As pointed above, *H -COL* is polynomial time solvable otherwise. Note that the classic k -colourability problem is a special case of the problem *H -COL* when H is K_k and the input graph G is irreflexive.

The problem of deciding the existence of a compaction to a fixed graph H , called the *compaction problem for H* , and denoted as *COMP- H* , asks whether or not an input graph G compacts to H .

The problem *COMP-H*, in general, where H is a fixed partially reflexive graph, can be viewed as the problem to decide whether or not it is possible to partition the vertices of a graph into certain fixed number of distinct non-empty sets such that there is at least one edge between some pair of distinct sets, and there is no edge between all other pair of distinct sets, and certain sets may be required to be independent (an independent set has no edge), where the sets and edges correspond to the vertices and edges of H , and an independent set in particular correspond to a vertex without a loop in H .

When both G and H are input graphs (i.e., H is not fixed), and H is reflexive, the problem of deciding whether or not G compacts to H has been studied in [Karabeg and Karabeg, 1991, 1993]. Some related work has recently been studied in [Feder, Hell, Klein, and Motwani, 1999, 2003]. Note that unlike the problem *H-COL*, the problem *COMP-H* is still interesting if H has a loop or H is bipartite.

The problem of deciding the existence of a retraction to a fixed graph H , called the *retraction problem for H* , and denoted as *RET-H*, asks whether or not an input graph G , containing H as an induced subgraph, retracts to H . Retraction problems have been of continuing interest in graph theory for a long time and have been studied in various literature including [Hell, 1972, 1974], [Nowakowski and Rival, 1979], [Pesch and Poguntke, 1985], [Bandelt, Dahlmann, and Schutte, 1987], [Hell and Rival, 1987], [Pesch, 1988], [Feder and Winkler, 1988], [Bandelt, Farber, and Hell, 1993], [Feder and Hell, 1998], [Feder, Hell, and Huang, 1999]. The problem *RET-H* can again be viewed as the partition problem, as described above, with the restriction that each vertex of H is in a distinct set of the partition.

Note that the graph H for the problems *H-COL*, *COMP-H*, and *RET-H* is assumed to be fixed by default even if not explicitly mentioned.

1.3 Motivation and Results

It is not difficult to show that for every fixed graph H , if *RET-H* is solvable in polynomial time then *COMP-H* is also solvable in polynomial time (a polynomial transformation from *COMP-H* to *RET-H* under Turing reduction is shown in [Vikas, 2004b]). Is the converse true? It was also asked, in the context of reflexive graphs, by Peter Winkler in 1988 (personal communication, cf. [Feder and Winkler, 1988]). Thus the question is whether *RET-H* and *COMP-H* are polynomially equivalent for every fixed graph H . The answer to this is not known even when H is reflexive or bipartite. However, it is shown in [Vikas, 2004b] that for every fixed reflexive (bipartite) graph H there exists a fixed reflexive (bipartite) graph H' such that *RET-H* and *COMP-H'* are polynomially equivalent.

Using the above result of [Vikas, 2004b], and results of [Feder and Hell, 1998] and [Feder and Vardi, 1998], it is established in [Vikas, 2004b] that for every constraint satisfaction problem Π (with fixed templates), there exists a fixed reflexive (bipartite) graph H such that the constraint satisfaction problem Π and the compaction problem *COMP-H* are polynomially equivalent. Since it is thought to be likely difficult to determine whether every constraint satisfaction problem (with fixed templates) is polynomial time solvable or NP-complete, thus

evidence is provided in [Vikas, 2004b] that it is likely to be difficult to determine whether for every fixed reflexive (bipartite) graph H , the problem $COMP-H$ is polynomial time solvable or NP-complete. Similar evidence has been shown for $RET-H$ in [Feder and Hell, 1998] in the case of fixed reflexive graphs H , and in [Feder and Vardi, 1998] in the case of fixed bipartite graphs H . Issues related to the constraint satisfaction problem have also been considered in [Feder and Vardi, 1993, 1998].

We however give in this paper, a complete complexity classification of $COMP-H$ and $RET-H$ when H has four or fewer vertices, i.e., for every graph H with at most four vertices (including when H is partially reflexive), we determine whether $COMP-H$ is polynomial time solvable or NP-complete, and whether $RET-H$ is polynomial time solvable or NP-complete. We find that the complexity classification of $COMP-H$ and $RET-H$ do not differ for such graphs H . Studying the complexity classification of the compaction and retraction problems for graphs with at most four vertices has an additional significance as it includes a widely publicised open problem posed by Peter Winkler in 1988 to determine the complexity of $COMP-H$ when H is a reflexive 4-cycle. This has been shown to be NP-complete in [Vikas, 1999, 2003]. The problem was asked in relation to the general problem mentioned earlier concerning the equivalence of the compaction and retraction problems, as the unique smallest reflexive graph H for which $RET-H$ is NP-complete turned out to be a reflexive 4-cycle.

Thus our study in this paper is motivated by two issues. One issue is concerned with the complete complexity classification of the compaction and retraction problems, and the other issue is concerned with the equivalence of the compaction and retraction problems. We present results in this paper resolving fully the two issues for graphs up to four vertices. We hope that the techniques and constructions developed in this paper and the papers [Vikas, 1999, 2003, 2004a, 2004b] would be helpful in resolving the bigger problem whether the compaction and retraction problems are equivalent for all graphs.

We have more results showing that for several graphs H , the problems $RET-H$ and $COMP-H$ are polynomially equivalent. We mention below a few classes of such graphs. We do not know of any graph H for which the complexity classification of $RET-H$ and $COMP-H$ differ.

It is known that $RET-H$ is NP-complete when H is a reflexive k -cycle, for all $k \geq 4$, cf. [Feder and Hell, 1998], G. MacGillivray, 1988 (personal communication), and for $k = 4$, [Feder and Winkler, 1988] also. It is shown in [Vikas, 1999, 2003] that $COMP-H$ is NP-complete when H is a reflexive k -cycle, for all $k \geq 4$. In particular, as mentioned above, for $k = 4$, this result in [Vikas, 1999, 2003] solves a widely publicised open problem posed by Peter Winkler in 1988. When H is a reflexive chordal graph (which includes a reflexive 3-cycle), the problem $RET-H$ is polynomial time solvable [Feder and Hell, 1998], and hence $COMP-H$ is also polynomial time solvable.

It is also known that $RET-H$ is NP-complete when H is an irreflexive even k -cycle, for all even $k \geq 6$, cf. [Feder, Hell, and Huang, 1999], G. MacGillivray, 1988 (personal communication). It is shown in [Vikas, 1999, 2004a] that $COMP-$

H is NP-complete when H is an irreflexive even k -cycle, for all even $k \geq 6$. This result in [Vikas, 1999, 2004a] also solves a problem that has been of interest, since a long time, to various people including Pavol Hell and Jaroslav Nesetril (personal communications). When H is a chordal bipartite graph (which includes an irreflexive 4-cycle), the problem $RET-H$ is polynomial time solvable [Bandelt, Dahlmann, and Schutte, 1987], and hence $COMP-H$ is also polynomial time solvable.

The case of irreflexive odd cycles is a special case of a more general result whereby $RET-H$ and $COMP-H$ are polynomially equivalent for every non-bipartite irreflexive graph H . It follows from the complexity of $H-COL$ [Hell and Nesetril, 1990] that $RET-H$ and $COMP-H$ are NP-complete for any non-bipartite irreflexive graph H ; in particular if H is an irreflexive odd k -cycle then $RET-H$ and $COMP-H$ are NP-complete, for all odd $k \geq 3$. Thus we conclude that $RET-H$ and $COMP-H$ both are NP-complete when H is an irreflexive k -cycle, for all $k \geq 3$, $k \neq 4$, and polynomial time solvable, for $k = 4$.

Let H be a graph and let V_L denote its set of vertices which have loops. It is shown by Feder, Hell, and Huang (personal communication, cf. [Feder, Hell, and Huang, 1999]) that if H is connected but V_L is not then $RET-H$ is NP-complete. It is also shown by them (cf. [Feder, Hell, and Huang, 1999]) that if V_L is connected and H is a tree then $RET-H$ is polynomial time solvable.

With regards to more general graphs (not necessarily with at most four vertices), we show in this paper that if H is a path of length ≥ 2 , with loops on the first (origin) and the last (terminus) vertices only, then $COMP-H$ is NP-complete. Using this result, we also show in this paper that $COMP-H$ is NP-complete when H is a graph consisting of only several internally disjoint paths of the same length ≥ 2 , having loops only on the common origin and terminus vertices. Also, we show in this paper that if H is a graph consisting of only a path of length ≥ 2 , and a vertex outside the path adjacent to every vertex of the path, with loops only on the origin and the terminus vertices of the path, then $COMP-H$ is NP-complete. We do not include the proofs here for these general graphs. For these graphs H , it follows from the above result of Feder, Hell, and Huang that $RET-H$ is also NP-complete.

We also show in this paper, the relationship between compaction to connected and disconnected graphs, and the relationship between retraction to connected and disconnected graphs. These results establish that the complexity of $COMP-H$ and $RET-H$ is not affected whether the graph H is connected or disconnected. Hence it would be sufficient to consider only connected graphs H when determining the complexity of $COMP-H$ or $RET-H$ in the next section. We do not include the proofs here for these results.

In the figures in this paper, we shall not be depicting any edge vh of G , with $v \in V(G - H)$ and $h \in V(H)$, where G is any graph containing H as an induced subgraph, i.e., G is an instance of $RET-H$. In Section 2, we give a complete complexity classification of $COMP-H$ as well as $RET-H$ when H has four or fewer vertices.

2 A Complete Complexity Classification of Compaction and Retraction to All Graphs with Four or Fewer Vertices

In this section, we give a complete complexity classification of *COMP-H* and *RET-H* when H has four or fewer vertices. We shall see that the complexity classification of *COMP-H* and *RET-H* do not differ for such graphs H . When considering graphs H with four or fewer vertices, we will not be separately considering graphs H that fall in the categories discussed in Section 1.3, or are trivial (we do not explicitly point out all such cases here). Also, due to our results on compaction and retraction to connected and disconnected graphs, we will not be separately considering graphs H that are disconnected. For graphs H with four or fewer vertices, this leaves us to consider the graphs H in Figure 1. We consider each of the graphs H in Figure 1, and present the complexity results for *COMP-H* and *RET-H*.

For the graphs H in Figures 1(a), (b), (c), (d), (e), (f), (j), (k), (m), and (p), the set of vertices of H with loops is disconnected (and H is connected), and hence it follows that *RET-H* is NP-complete, cf. [Feder, Hell, and Huang, 1999]; we however need to determine the complexity of *COMP-H*. We will be including here details for only some of the graphs.

Theorem 2.1 *COMP-H is NP-complete for the graph H in Figure 1(a).*

Proof. Let H be the path $h_0h_1h_2$ in Figure 1(a) with loops on h_0 and h_2 only. It is clear that *COMP-H* is in NP. We prove NP-completeness of *COMP-H* by giving a polynomial transformation from *RET-H* to *COMP-H*. As mentioned above, the problem *RET-H* is NP-complete, cf. [Feder, Hell, and Huang, 1999]. Let G be a graph containing H as an induced subgraph, i.e., let G be an instance of *RET-H*. We construct in time polynomial in the size of G , a graph G' (containing G as an induced subgraph) such that the following statements (i), (ii), and (iii) are equivalent :

- (i) G retracts to H .
- (ii) G' retracts to H .
- (iii) G' compacts to H .

We prove that (i) is equivalent to (ii), and (ii) is equivalent to (iii), in Lemma 2.1.1 and Lemma 2.1.2 respectively. Since *RET-H* is NP-complete, this shows that *COMP-H* is NP-complete.

The construction of G' is as follows. For each vertex v in $V(G - H)$, we add to G three distinct new vertices : u_v adjacent to v , h_0 ; w_v adjacent to v , u_v ; and y_v adjacent to u_v , w_v , h_2 . Thus u_v , w_v , and y_v form a triangle, and u_v , w_v , and v form a triangle. See Figure 2. Note that there could be edges in G from v to some vertices of H but as mentioned earlier, in Figure 2 and all subsequent figures in this paper, we are not depicting these edges. This completes the construction of G' .

We now prove the following two lemmas in order to prove the theorem.

Lemma 2.1.1 *G retracts to H if and only if G' retracts to H .*

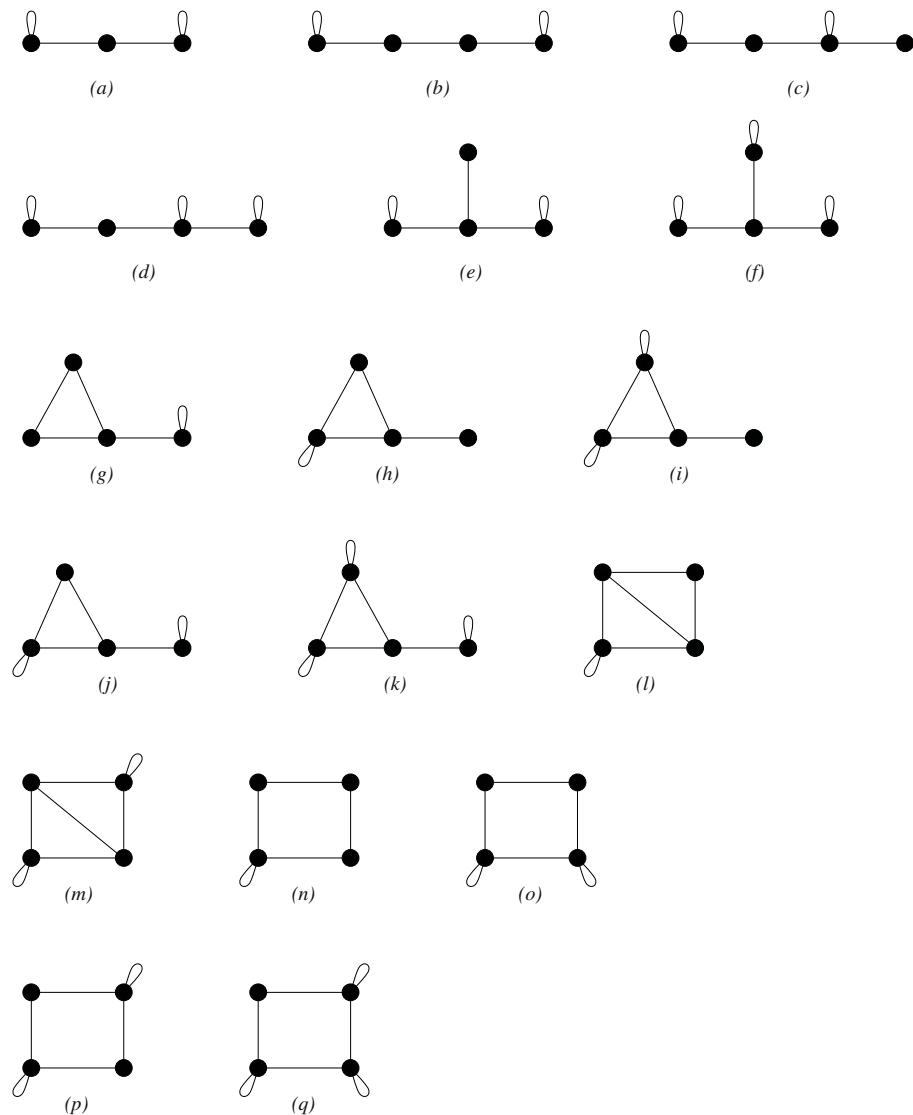


Fig. 1. List of graphs H to be considered

Proof. If G' retracts to H then it is clear that G retracts to H since G is a subgraph of G' . Now suppose that $r : G \rightarrow H$ is a retraction. We define a retraction $r' : G' \rightarrow H$ as follows.

For each vertex v of the graph G , we define

$$r'(v) = r(v).$$

For the vertices u_v, w_v , and y_v of G' , with $v \in V(G - H)$, we define

$$r'(u_v) = h_0, r'(w_v) = h_0, \text{ and } r'(y_v) = h_1, \text{ if } r(v) = h_0 \text{ or } h_1, \text{ and} \\ r'(u_v) = h_1, r'(w_v) = h_2, \text{ and } r'(y_v) = h_2, \text{ if } r(v) = h_2.$$

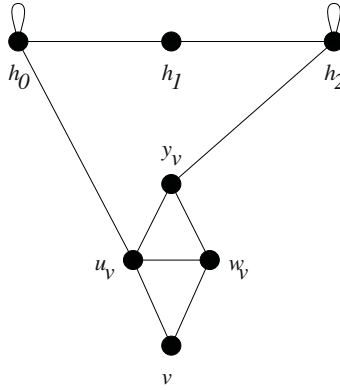


Fig. 2. Construction of G' for a vertex v in $G - H$

It is not difficult to see that for every edge ab of G' , $r'(a)r'(b)$ is an edge of H , i.e., $r' : G' \rightarrow H$ is a homomorphism. Recall that the edges of G' are : gg' , vu_v , vw_v , h_0u_v , h_2y_v , u_vw_v , u_vy_v , and w_vy_v , with $gg' \in E(G)$, $v \in V(G - H)$. Since $r'(h) = r(h) = h$, for all $h \in V(H)$, the homomorphism $r' : G' \rightarrow H$ is a retraction, and the lemma is proved. \square

Lemma 2.1.2 G' retracts to H if and only if G' compacts to H .

Proof. If G' retracts to H then by definition G' compacts to H . Now suppose that $c : G' \rightarrow H$ is a compaction. We first prove that $c(h_0) \neq c(h_2)$. Suppose that $c(h_0) = c(h_2)$. Since h_0 and h_2 both have loops, it must be that $c(h_0) = c(h_2) = h_0$ or $c(h_0) = c(h_2) = h_2$. Without loss of generality, let $c(h_0) = c(h_2) = h_0$. We can assume this, as due to symmetry of vertices in H , we can always redefine the compaction c so that $c(h_0) = c(h_2) = h_0$. Since u_v , y_v , and h_1 are adjacent to h_0 or h_2 , $c(u_v) \neq h_2$, $c(y_v) \neq h_2$, and $c(h_1) \neq h_2$, with $v \in V(G - H)$. This implies that if $c(w_v) = h_2$, for some vertex $v \in V(G - H)$, then $c(u_v) = c(y_v) = h_1$, as u_v and y_v are adjacent to w_v . But this is impossible as u_v is adjacent to y_v , and h_1 does not have a loop. Hence $c(w_v) \neq h_2$, for any $v \in V(G - H)$. Similarly, if $c(v) = h_2$, for some vertex $v \in V(G - H)$, then $c(u_v) = c(w_v) = h_1$, and h_1 does not have a loop. Hence $c(v) \neq h_2$, for any $v \in V(G - H)$.

Thus, we have shown that $c(a) \neq h_2$, for all $a \in V(G')$, contradicting our assumption that $c : G' \rightarrow H$ is a compaction. Hence, it must be that $c(h_0) \neq c(h_2)$. Since h_0 and h_2 are the only vertices in H with a loop, $c(\{h_0, h_2\}) = \{h_0, h_2\}$. Without loss of generality, let $c(h_0) = h_0$ and $c(h_2) = h_2$ (due to symmetry). Since h_1 is adjacent to h_0 and h_2 , we have $c(h_1) = h_1$. Thus $c : G' \rightarrow H$ is a retraction, and the lemma is proved. \square

This completes the proof of Theorem 2.1. \square

Theorem 2.2 *COMP-H* is NP-complete for the graph H in Figure 1(g).

Proof. Clearly, $COMP-H$ is in NP. Let S be the irreflexive triangle $h_0h_1h_2h_0$. We give a polynomial transformation from $RET-S$ to $COMP-H$. As mentioned earlier, it follows from [Hell and Nesetril, 1990] that $RET-S$ is NP-complete. Let a graph G containing S as an induced subgraph be an instance of $RET-S$. We construct in time polynomial in the size of G , a graph G' (containing G and H as induced subgraphs) such that the following statements (i), (ii), and (iii) are equivalent : (i) G retracts to S , (ii) G' retracts to H , (iii) G' compacts to H . We prove that (i) is equivalent to (ii), and (ii) is equivalent to (iii), in two separate lemmas, which we do not include here. Since $RET-S$ is NP-complete, this shows that $COMP-H$ is NP-complete. We note that the equivalence of (i) and (ii) shows that $RET-H$ is also NP-complete. The fact that $RET-H$ is NP-complete, we have also shown using a simpler construction which we do not include here. The complexity of $RET-H$ was also unknown before.

The construction of G' is as follows. For each vertex v of $G - S$, we add to G six distinct new vertices : u_v adjacent to v and h_0 ; w_v adjacent to v , h_1 , and u_v ; x_v adjacent to v and h_0 ; y_v adjacent to x_v , w_v , and h_0 ; z_v adjacent to v and h_1 ; and s_v adjacent to s_v , u_v , and h_1 . We also add to G a new vertex h_3 adjacent to h_2 and itself only (thus H is an induced subgraph of the resulting graph). See Figure 3. This completes the construction of G' . \square

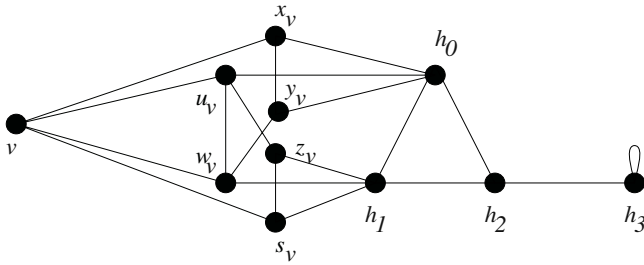


Fig. 3. Construction of G' for a vertex v in $G - H$

Theorem 2.3 $COMP-H$ and $RET-H$ are polynomial time solvable for the graph H in Figure 1(q).

Proof. Let the graph H in Figure 1(q) be $h_0h_1h_2h_3h_0$ with loops on all the vertices except h_3 . We give a polynomial time algorithm for $RET-H$. Since $COMP-H$ polynomially transforms to $RET-H$ (under Turing reduction), this shows that $COMP-H$ is also polynomial time solvable. Let a graph G containing H as an induced subgraph be an instance of $RET-H$. Let S be a subgraph of G induced by $Nbr(h_3)$ (which includes h_0 and h_2). The algorithm for $RET-H$ is described below.

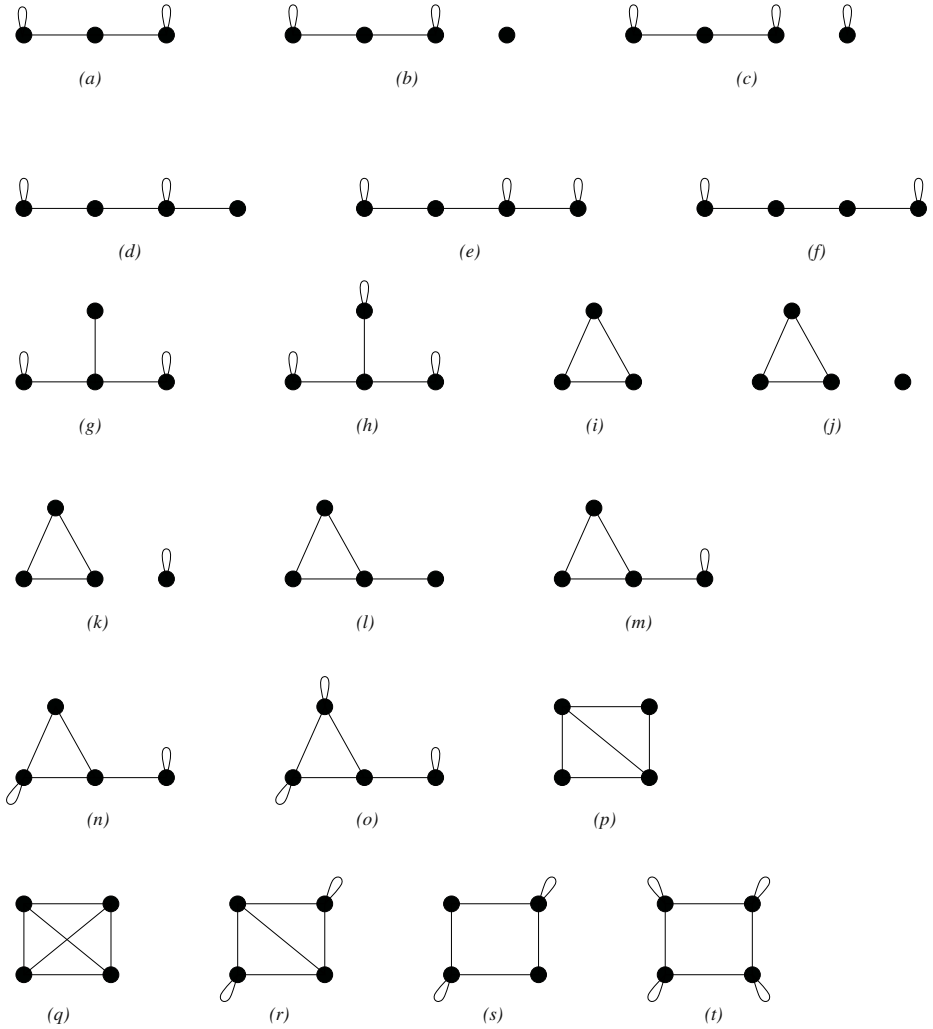


Fig. 4. List of All Graphs H with at most Four Vertices for which $COMP-H$ and $RET-H$ are NP-complete

1. If there exists a path from h_0 to h_2 in S then G does not retract to H .
2. Else G retracts to H with a retraction $r : G \rightarrow H$ defined as follows. We have that h_0 and h_2 are disconnected in S . Let Z be a component of S containing h_2 . Thus $h_0 \notin Z$. We define

$$\begin{aligned}
 r(h) &= h, \text{ for all } h \in V(H), \\
 r(v) &= h_0, \text{ for all } v \in Nbr(h_3) - V(Z), \\
 r(v) &= h_2, \text{ for all } v \in V(Z), \text{ and} \\
 r(v) &= h_1, \text{ for all } v \in V(G) - (Nbr(h_3) \cup V(H)).
 \end{aligned}$$

□

We consider the other graphs also, and show that for all the graphs H in Figure 1, *COMP-H* and *RET-H* are NP-complete or polynomial time solvable, and that the complexity classification of *COMP-H* and *RET-H* do not differ. For the graphs H in Figures 1(a), (b), (c), (d), (e), (f), (g), (j), (k), (m), and (p), both *COMP-H* and *RET-H* are NP-complete. For the graphs H in Figures 1(h), (i), (l), (n), (o), and (q), both *COMP-H* and *RET-H* are polynomial time solvable (it is not difficult to see that none of these graphs are absolute retracts which would imply polynomiality anyway; we do not discuss absolute retracts here). As discussed before, for the graphs H with at most four vertices not listed in Figure 1, either we know from the results mentioned in Section 1.3, or are trivial, or we can infer from our results on compaction and retraction to connected and disconnected graphs, that *COMP-H* and *RET-H* are NP-complete or polynomial time solvable, and notice again that the complexity classification of *COMP-H* and *RET-H* do not differ.

A list of all graphs H with at most four vertices for which *COMP-H* and *RET-H* are NP-complete is given in Figure 4. For all other graphs H with at most four vertices, not in this list, both *COMP-H* and *RET-H* are polynomial time solvable.

References

1. H. J. Bandelt, A. Dahlmann, and H. Schutte, Absolute Retracts of Bipartite Graphs, *Discrete Applied Mathematics*, 16, 191-215, 1987.
2. H. J. Bandelt, M. Farber, and P. Hell, Absolute Reflexive Retracts and Absolute Bipartite Retracts, *Discrete Applied Mathematics*, 44, 9-20, 1993.
3. T. Feder and P. Hell, List Homomorphisms to Reflexive Graphs, *Journal of Combinatorial Theory, Series B*, 72, 236-250, 1998.
4. T. Feder, P. Hell, and J. Huang, List Homomorphisms and Circular Arc Graphs, *Combinatorica*, Volume 19, Number 4, 487-505, 1999.
5. T. Feder, P. Hell, S. Klein, and R. Motwani, Complexity of Graph Partition Problems, in *Proceedings of the 31st Annual ACM Symposium on Theory of Computing (STOC)*, Atlanta, Georgia, 1999.
6. T. Feder, P. Hell, S. Klein, and R. Motwani, List Partitions, *SIAM Journal on Discrete Mathematics*, 16, 449-478, 2003.
7. T. Feder and M. Y. Vardi, Monotone Monadic SNP and Constraint Satisfaction, in *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, San Diego, California, 1993.
8. T. Feder and M. Y. Vardi, The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction : A Study through Datalog and Group Theory, *SIAM Journal on Computing*, 28, 57-104, 1998.
9. T. Feder and P. Winkler, manuscript, 1988.
10. F. Harary, *Graph Theory*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1969.
11. P. Hell, *Retractions de Graphes*, Ph.D. Thesis, Universite de Montreal, 1972.
12. P. Hell, Retracts in Graphs, in *Graphs and Combinatorics*, Lecture Notes in Mathematics, Springer-Verlag, Berlin, 406, 291-301, 1974.
13. P. Hell and D. J. Miller, Graphs with Forbidden Homomorphic Images, *Annals of the New York Academy of Sciences*, 319, 270-280, 1979.

14. P. Hell and J. Nešetřil, On the Complexity of H -colouring, *Journal of Combinatorial Theory, Series B*, 48, 92-110, 1990.
15. P. Hell and I. Rival, Absolute Retracts and Varieties of Reflexive Graphs, *Canadian Journal of Mathematics*, 39, 544-567, 1987.
16. A. Karabeg and D. Karabeg, Graph Compaction, *Graph Theory Notes of New York XXI*, The New York Academy of Sciences, 44-51, 1991.
17. A. Karabeg and D. Karabeg, Graph Compaction, manuscript, 1993.
18. R. Nowakowski and I. Rival, Fixed-Edge Theorem for Graphs with Loops, *Journal of Graph Theory*, 3, 339-350, 1979.
19. E. Pesch, Retracts of Graphs, *Mathematical Systems in Economics*, 110, Athenaum, Frankfurt am Main, 1988.
20. E. Pesch and W. Poguntke, A Characterization of Absolute Retracts of n -Chromatic Graphs, *Discrete Mathematics*, 57, 99-104, 1985.
21. N. Vikas, Computational Complexity of Compaction to Cycles, in *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Baltimore, Maryland, 1999.
22. N. Vikas, Computational Complexity of Compaction to Reflexive Cycles, *SIAM Journal on Computing*, 32, 253-280, 2003.
23. N. Vikas, Computational Complexity of Compaction to Irreflexive Cycles, *Journal of Computer and System Sciences*, 68, 473-496, 2004a.
24. N. Vikas, Compaction, Retraction, and Constraint Satisfaction, *SIAM Journal on Computing*, 33, 761-782, 2004b.

One-to-Many Disjoint Path Covers in a Graph with Faulty Elements*

Jung-Heum Park

School of Computer Science and Information Engineering
The Catholic University of Korea, Puchon, Kyonggi-do 420-743, Korea
j.h.park@catholic.ac.kr

Abstract. In a graph G , k disjoint paths joining a single source and k distinct sinks that cover all the vertices in the graph are called a *one-to-many k -disjoint path cover* of G . We consider a k -disjoint path cover in a graph with faulty vertices and/or edges obtained by merging two graphs H_0 and H_1 , $|V(H_0)| = |V(H_1)| = n$, with n pairwise nonadjacent edges joining vertices in H_0 and vertices in H_1 . We present a sufficient condition for such a graph to have a k -disjoint path cover and give the construction scheme. Applying our main result to interconnection graphs, we observe that when there are f or less faulty elements, all of recursive circulant $G(2^m, 4)$, twisted cube TQ_m , and crossed cube CQ_m of degree m have k -disjoint path covers for any $f \geq 0$ and $k \geq 2$ such that $f + k \leq m - 1$.

1 Introduction

Usually a measure of reliability (or fault-tolerance) of an interconnection network is given by the maximum number of nodes which can fail simultaneously without prohibiting the ability of each fault-free node to communicate with all other fault-free nodes. Connectivity of an interconnection graph corresponds to the reliability of the interconnection network which is subject to node failures.

It is well-known that connectivity of a graph G was characterized in terms of disjoint paths joining a pair of vertices in G . According to Menger's theorem, a graph G is k -connected if and only if for every pair of vertices s and t , G has k disjoint paths (of type one-to-one) joining them. It is straightforward to verify that a graph G is k -connected if and only if G has k disjoint paths (of type one-to-many) joining every source s and k distinct sinks t_1, t_2, \dots, t_k such that $t_i \neq s$ for all $1 \leq i \leq k$.

Sometimes "restricted" one-to-many disjoint paths joining a source s and k distinct sinks t_1, t_2, \dots, t_k are required. When we analyze Rabin number[1] of a graph G , we are to find k disjoint paths P_1, P_2, \dots, P_k such that $\max\{l_i\}$ is as small as possible, where P_i is an s - t_i path of length l_i , $1 \leq i \leq k$. A star-like tree[2] is a subdivision of a star, a tree with only one vertex which is not an endvertex. Given a source s and lengths l_1, l_2, \dots, l_k of k paths, the star-like tree problem is essentially to find one-to-many disjoint paths joining s and some k sinks whose lengths are l_1, l_2, \dots, l_k , respectively.

* This work was supported by grant No. R05-2003-000-11506-0 from the Basic Research Program of the Korea Science & Engineering Foundation.

Given a source s and a set of k sinks $T = \{t_1, t_2, \dots, t_k\}$ in a graph G , we are concerned with one-to-many disjoint paths P_1, P_2, \dots, P_k in G joining s and T that *cover* all the vertices in the graph, that is, $\bigcup_{1 \leq i \leq k} V(P_i) = V(G)$ and $V(P_i) \cap V(P_j) = \{s\}$ for all $i \neq j$. Here $V(P_i)$ and $V(G)$ denote the vertex sets of P_i and G , respectively. We call such k disjoint paths a *one-to-many k -disjoint path cover* (in short, *one-to-many k -DPC*) of G .

Embedding of linear arrays and rings into a faulty interconnection graph is one of the central issues in parallel processing. The problem is modelled as finding as long fault-free paths and cycles as possible in the graph with some faulty vertices and/or edges. A graph G is called *f -fault hamiltonian* (resp. *f -fault hamiltonian-connected*) if there exists a hamiltonian cycle (resp. if each pair of vertices are joined by a hamiltonian path) in $G \setminus F$ for any set F of faulty elements such that $|F| \leq f$. For a graph G to be *f -fault hamiltonian* (resp. *f -fault hamiltonian-connected*), it is necessary that $f \leq \delta(G) - 2$ (resp. $f \leq \delta(G) - 3$), where $\delta(G)$ is the minimum degree of G .

To a graph G with a set of faulty elements F , the definition of a one-to-many disjoint path cover can be extended. Given a source s and a set of k sinks $T = \{t_1, t_2, \dots, t_k\}$ in $G \setminus F$, a one-to-many k -disjoint path cover joining s and T is k disjoint paths P_i joining s and t_i , $1 \leq i \leq k$, such that $\bigcup_{1 \leq i \leq k} V(P_i) = V(G) \setminus F$, $V(P_i) \cap V(P_j) = \{s\}$ for all $i \neq j$, and every edge on each path P_i is fault-free. Such a one-to-many k -DPC is denoted by $k\text{-DPC}[s, \{t_1, t_2, \dots, t_k\} | G, F]$. A graph G is called *f -fault one-to-many k -disjoint path coverable* if for any set F of faulty elements such that $|F| \leq f$, G has $k\text{-DPC}[s, \{t_1, t_2, \dots, t_k\} | G, F]$ for every source s and k distinct sinks t_1, t_2, \dots, t_k in $G \setminus F$.

Proposition 1. *Let G be a graph. The following statements are equivalent.*

- (a) G is f -fault hamiltonian-connected.
- (b) G is f -fault one-to-many 1-disjoint path coverable.
- (c) G is f -fault one-to-many 2-disjoint path coverable.

We are given two graphs H_0 and H_1 with n vertices. We denote by V_i and E_i the vertex set and edge set of H_i , $i = 0, 1$, respectively. We let $V_0 = \{v_1, v_2, \dots, v_n\}$ and $V_1 = \{w_1, w_2, \dots, w_n\}$. With respect to a permutation $M = (i_1, i_2, \dots, i_n)$ of $\{1, 2, \dots, n\}$, we can “merge” the two graphs into a graph $H_0 \oplus_M H_1$ with $2n$ vertices in such a way that the vertex set $V = V_0 \cup V_1$ and the edge set $E = E_0 \cup E_1 \cup E_2$, where $E_2 = \{(v_j, w_{i_j}) | 1 \leq j \leq n\}$. We denote by $H_0 \oplus H_1$ an arbitrary graph obtained by merging H_0 and H_1 w.r.t. some permutation M . Here, H_0 and H_1 are called *components* of $H_0 \oplus H_1$.

In this paper, we will show that we can always construct f -fault one-to-many $k + 1$ -DPC in $H_0 \oplus H_1$ by using f -fault one-to-many k -DPC of H_i and fault-hamiltonicity of H_i , $i = 0, 1$. Precisely speaking, we will prove the following.

Theorem 1. *For $f \geq 0$ and $k \geq 2$, let H_i , $i = 0, 1$, be a graph with n vertices satisfying the following three conditions:*

- (a) H_i is f -fault one-to-many k -disjoint path coverable.
- (b) H_i is $f + k - 2$ -fault hamiltonian-connected (2-disjoint path coverable).
- (c) H_i is $f + k - 1$ -fault hamiltonian.

Then, $H_0 \oplus H_1$ is f -fault one-to-many $k + 1$ -disjoint path coverable.

By applying the above theorem to interconnection graphs, we will show that all of recursive circulant $G(2^m, 4)$, twisted cube TQ_m , and crossed cube CQ_m of degree m are f -fault one-to-many k -disjoint path coverable for every pair of $f \geq 0$ and $k \geq 2$ such that $f + k \leq m - 1$.

2 Construction of One-to-Many DPC in $H_0 \oplus H_1$

In this section, we will prove Theorem 1 by constructing a one-to-many $k + 1$ -DPC in $H_0 \oplus H_1$ when the number of faulty elements is f or less. Since each H_i satisfies the condition (c) of Theorem 1, it is necessary that $f + k - 1 \leq \delta_i - 2$, where $\delta_i = \delta(H_i)$. That is,

$$f + k \leq \delta - 1,$$

where $\delta = \min\{\delta_0, \delta_1\}$. We assume w.l.o.g. that the source s is in H_0 , and there are $k + 1$ distinct sinks $T = \{t_1, t_2, \dots, t_{k+1}\}$, k_i sinks in H_i , $i = 0, 1$. Thus,

$$k_0 + k_1 = k + 1.$$

We assume that $\{t_1, t_2, \dots, t_{k+1}\}$ is the set of sinks in H_1 . We denote by F the set of faulty elements in $H_0 \oplus H_1$. We let f_0 , f_1 , and f_2 be the numbers of faulty elements in H_0 , in H_1 , and between H_0 and H_1 , respectively. Let f_i^v and f_i^e be the numbers of faulty vertices and edges in H_i , respectively. Then,

$$|F| = f_0 + f_1 + f_2 \leq f, \text{ and } f_i^v + f_i^e = f_i, i = 0, 1.$$

Before going on, we need some definitions and notation. For a vertex v in $H_0 \oplus H_1$, we denote by \bar{v} the vertex adjacent to v which is in a different component from v . A vertex v in $H_0 \oplus H_1$ is called *free* if $v \neq s$, $v \notin T$, and $v \notin F$. A *free bridge* of a vertex w is the path (w, \bar{w}) of length one if $\bar{w} \notin T$, $\bar{w} \notin F$, and $(w, \bar{w}) \notin F$; otherwise, it is a path (w, v, \bar{v}) of length two such that $v \neq \bar{w}$, v and \bar{v} are free vertices, and $(w, v), (v, \bar{v}) \notin F$.

Lemma 1. *For any source and sink w in $H_0 \oplus H_1$, there exists a free bridge of w .*

Proof. There are at least $\delta + 1$ candidates for a free bridge of w , and at most $f + k + 1$ elements “block” the candidates. For the source w , at most f faulty elements and $k + 1$ sinks form blocking elements. For a sink w , blocking elements are faulty elements, k sinks (excluding w itself), and the source. Since $f + k \leq \delta - 1$, the lemma is proved. \square

We define L to be a set of edges joining a sink w in H_1 and \bar{w} in H_0 different from s such that (w, \bar{w}) is not the free bridge of w . That is,

$$L = \{(w, \bar{w}) | w \text{ is a sink in } H_1, \bar{w} \neq s, \text{ and } \bar{w} \in T \text{ or } \bar{w} \in F \text{ or } (w, \bar{w}) \in F\}.$$

An edge (w, \bar{w}) in L is called *matched* and its endvertices are called *matched* vertices. We denote by f_2^L be the number of matched faulty edges, faulty edges in L . We let $l = |L|$. Then,

$$l \leq k_0 + f_0^v + f_2^L.$$

If $\bar{s} \in T$ and $(s, \bar{s}) \in F$, we let $\alpha = 1$; otherwise, let $\alpha = 0$. Obviously,

$$l + \alpha \leq k_1, \text{ and } f_2^L + \alpha \leq f_2.$$

First we will construct f -fault one-to-many $k+1$ -DPC for the case of $k_1 \geq 1$, and then we will consider the case of $k_1 = 0$ later in Section 2.3. Construction I gives an f -fault one-to-many $k+1$ -DPC unless the following conditions of C1, C2, and C3 are satisfied:

C1: $k_1 = 1$, $\bar{s} \in T$, $f_0 = f$,

C2: $k_1 + f_1 \geq k + f$,

C3: $l + \alpha \geq 2$, $l = k_0 + f_0^v + f_2^L$, $f = f_0^v + f_1 + f_2^L + \alpha$.

Construction II considers the case that satisfies C3. Remaining cases are considered later in Section 2.1 and 2.2. We denote by $H[v, w|H_i, F]$ a hamiltonian path in $H_i \setminus F$ joining a pair of fault-free vertices v and w . We let $F_i = F \cap V_i$.

Construction I

UNDER the condition of $k_1 \geq 1$.

UNLESS C1, C2, and C3

1. Let $t_1 = \begin{cases} \text{an arbitrary matched sink, if } l \geq 1; \\ \bar{s}, & \text{if } l = 0 \text{ and } \bar{s} \in T; \\ \text{an arbitrary sink,} & \text{if } l = 0 \text{ and } \bar{s} \notin T. \end{cases}$
2. Find a free bridge of s and free bridges $B_j = (t_j, \dots, t'_j)$ of t_j for all j , $2 \leq j \leq k_1$. They should be pairwise disjoint.
3. When (s, \bar{s}) is the free bridge of s :
 - Find k -DPC $[s, \{t'_2, \dots, t'_{k_1}, t_{k_1+1}, \dots, t_{k+1}\} | H_0, F_0]$.
 - Find $H[\bar{s}, t_1 | H_1, F_1 \cup F']$, where $F' = \bigcup_{2 \leq j \leq k_1} \{V(B_j) \cap V_1\}$.
 - Merge the k -DPC and the hamiltonian path obtained in the previous step with the edge (s, \bar{s}) and the free bridges B_j , $2 \leq j \leq k_1$.
4. When (s, \bar{s}) is not the free bridge of s : Let (s, x, y) be the free bridge of s .
 - 4.1 When $(s, \bar{s}) \notin F$, $\bar{s} \in T$, and $k_1 \geq 2$: Let z be t_1 if $\bar{s} \neq t_1$; otherwise, let z be t_2 . If $\bar{s} \neq t_1$, we assume w.l.o.g. that $\bar{s} = t_2$.
 - Find k -DPC $[s, \{t'_3, \dots, t'_{k_1}, t_{k_1+1}, \dots, t_{k+1}\} \cup \{x\} | H_0, F_0]$.
 - Find $H[y, z | H_1, F_1 \cup F']$, where $F' = \{\bar{s}\} \cup \bigcup_{3 \leq j \leq k_1} \{V(B_j) \cap V_1\}$.
 - Merge the k -DPC and the hamiltonian path obtained with the edges (s, \bar{s}) , (x, y) and the free bridges B_j , $3 \leq j \leq k_1$.
 - 4.2 When $(s, \bar{s}) \notin F$, $\bar{s} \in T$, and $k_1 = 1$, or $(s, \bar{s}) \in F$, or $\bar{s} \in F$: In this case, it holds true that $f_0 < f$. (Recall C1.)
 - Find k -DPC $[s, \{t'_2, \dots, t'_{k_1}, t_{k_1+1}, \dots, t_{k+1}\} | H_0, F_0 \cup \{x\}]$.
 - Find $H[y, t_1 | H_1, F_1 \cup F']$, where $F' = \bigcup_{2 \leq j \leq k_1} \{V(B_j) \cap V_1\}$.
 - Merge the k -DPC and the hamiltonian path obtained with the edge (x, y) and the free bridges B_j , $2 \leq j \leq k_1$.

When we find a k -DPC or a hamiltonian path, sometimes we regard some fault-free vertices and/or edges as faulty elements. They are called *virtual faults*. For example, in step 4.2 of Construction I, the vertex x is a virtual fault when we find a k -DPC, and F' is a set of virtual faults when we find a hamiltonian path.

The existence of pairwise disjoint free bridges in step 2 of Construction I can be shown by extending Lemma 1. The proof is omitted in this paper.

Lemma 2. *We can always find k -DPC's of H_0 in step 3, 4.1, and 4.2 of Construction I.*

Proof. In each step, the number of sinks is k and the number of faulty elements including virtual faults is at most f . The k -DPC's can be found since H_0 is f -fault one-to-many k -disjoint path coverable. \square

Lemma 3. *We can always find hamiltonian paths of H_1 in step 3, 4.1, and 4.2 of Construction I unless C2 and C3.*

Proof. Let $f' = |F'|$. A sink t_j in H_1 contributes to f' by 2 if t_j is matched or if $t_j = \bar{s}$ and $(s, \bar{s}) \in F$. For $l \geq 1$, $f' + f_1 = (k_1 - l) + 2(l - 1) + \alpha + f_1 = k_1 + l + \alpha + f_1 - 2$. When $l + \alpha \geq 2$, $f' + f_1 \leq k_1 + (k_0 + f_0^v + f_2^L) + \alpha + f_1 - 2 \leq (k + 1) + f - 2 \leq f + k - 1$. The equality holds only when $l = k_0 + f_0^v + f_2^L$ and $f = f_0^v + f_2^L + \alpha + f_1$. Unless C3, $f' + f_1 \leq f + k - 2$, and thus there exists a hamiltonian path between every pair of fault-free vertices in H_1 . When $l + \alpha < 2$, that is, $l = 1$ and $\alpha = 0$, $f' + f_1 = k_1 + 1 + f_1 - 2$. Unless C2, $f' + f_1 \leq f + k - 2$. For $l = 0$, $f' + f_1 \leq (k_1 - 1) + f_1$. Also in this case, unless C2, $f' + f_1 \leq f + k - 2$. This completes the proof. \square

Let us consider the case that satisfies C3. Obviously, $k_1 \geq 2$. Observe that for every free vertex $w (\neq \bar{s})$ in H_1 , (w, \bar{w}) is a free bridge of w .

Construction II

UNDER the condition C3: $l + \alpha \geq 2$, $l = k_0 + f_0^v + f_2^L$, $f = f_0^v + f_1 + f_2^L + \alpha$.

1. Let t_1 be a sink as defined in step 1 of Construction I.
2. Find a free bridge of s and free bridges $B_j = (t_j, \dots, t'_j)$ of t_j for all j , $2 \leq j \leq k_1$. They should be pairwise disjoint.
3. When (s, \bar{s}) is the free bridge of s ($\alpha = 0$, $l \geq 2$): We assume that t_2 is a matched sink.
 - Find $H[\bar{s}, t_1 | H_1, F_1 \cup F']$, where $F' = \{(t_1, t_2)\} \cup \bigcup_{3 \leq j \leq k_1} \{V(B_j) \cap V_1\}$. Let the hamiltonian path $(\bar{s}, \dots, t_2, z, \dots, t_1)$. Note that z is a free vertex.
 - Find k -DPC $[s, \{t'_3, \dots, t'_{k_1}, t_{k_1+1}, \dots, t_{k+1}\} \cup \{\bar{z}\} | H_0, F_0]$.
 - Merge the k -DPC and the hamiltonian path with the edges (s, \bar{s}) , (\bar{z}, z) and the free bridges B_j , $3 \leq j \leq k_1$.
4. When (s, \bar{s}) is not the free bridge of s : Let (s, x, y) be the free bridge of s .
 - 4.1 When $(s, \bar{s}) \notin F$ and $\bar{s} \in T$ ($\alpha = 0$, $l \geq 2$, $k_1 \geq 3$): We assume that t_2 is a matched sink and t_3 is \bar{s} .
 - Find $H[y, t_1 | H_1, F_1 \cup F']$, where $F' = \{(t_1, t_2)\} \cup \bigcup_{3 \leq j \leq k_1} \{V(B_j) \cap V_1\}$. Let the hamiltonian path $(y, \dots, t_2, z, \dots, t_1)$.
 - Find k -DPC $[s, \{t'_4, \dots, t'_{k_1}, t_{k_1+1}, \dots, t_{k+1}\} \cup \{x, \bar{z}\} | H_0, F_0]$.
 - Merge the k -DPC and the hamiltonian path with the edges (s, \bar{s}) , (x, y) , (\bar{z}, z) and the free bridges B_j , $4 \leq j \leq k_1$.

- 4.2 When $(s, \bar{s}) \in F$ or $\bar{s} \in F$: Let t_2 be a matched sink if $\bar{s} \notin T$; otherwise, let t_2 be \bar{s} . Note that $f_0 < f$.
- Find $H[y, t_1 | H_1, F_1 \cup F']$, where $F' = \{(t_1, t_2)\} \cup \bigcup_{3 \leq j \leq k_1} \{V(B_j) \cap V_1\}$. Let the hamiltonian path $(y, \dots, t_2, z, \dots, t_1)$.
 - Find k -DPC $[s, \{t'_3, \dots, t'_{k_1}, t_{k_1+1}, \dots, t_{k+1}\} \cup \{\bar{z}\} | H_0, F_0 \cup \{x\}]$.
 - Merge the k -DPC and the hamiltonian path with the edge (x, y) , (\bar{z}, z) and the free bridges B_j , $3 \leq j \leq k_1$.

It is easy to observe the existence of k -DPC's of H_0 in step 3, 4.1, and 4.2 of Construction II.

Lemma 4. *We can always find hamiltonian paths of H_1 in step 3, 4.1, and 4.2 of Construction II.*

Proof. Let $f' = |F'|$. For $\alpha = 0$ ($l \geq 2$), $f' + f_1 = (k_1 - l) + 2(l - 2) + 1 + f_1 = k_1 + l + f_1 - 3 = k_1 + (k_0 + f_0^v + f_2^L) + f_1 - 3 \leq (k + 1) + f - 3 = f + k - 2$. For $\alpha = 1$, $f' + f_1 = (k_1 - l - 1) + 2(l - 1) + 1 + f_1 = k_1 + l + f_1 - 2 = k_1 + (k_0 + f_0^v + f_2^L) + f_1 - 2 = (k + 1) + (f - 1) - 2 = f + k - 2$. This completes the proof. \square

Hereafter in this section, we will construct f -fault one-to-many $k + 1$ -DPC's for two exceptional cases of C1 and C2 and for the case of $k_1 = 0$.

2.1 When C2: $k_1 + f_1 \geq k + f$

The condition C2 is equivalent to that $k_0 + (f_0 + f_2) \leq 1$. When (s, \bar{s}) is not the free bridge of s , we denote by (s, x, y) the free bridge of s . Let v be the endvertex of the free bridge of s in H_1 . Of course, $v = \bar{s}$ if (s, \bar{s}) is the free bridge of s ; otherwise, $v = y$. We let

$$F' = \begin{cases} \emptyset, & \text{if } (s, \bar{s}) \text{ is the free bridge of } s; \\ \{x\}, & \text{if at least one of } (s, \bar{s}) \text{ and } \bar{s} \text{ are faulty.} \end{cases}$$

Case 1: $f_0 + f_2 = 0$ and $k_0 = 0$.

We first consider the case that \bar{s} is a free vertex or a faulty vertex. We find $H[v, t_1 | H_1, F_1]$ and let the path be $(v, \dots, t_{k+1}, z_{k+1}, \dots, t_k, z_k, \dots, t_2, z_2, \dots, t_1)$. We find k -DPC $[s, \{\bar{z}_2, \dots, \bar{z}_{k+1}\} | H_0, F']$, and then merge the k -DPC and the hamiltonian path with the free bridge of s and the edges (\bar{z}_j, z_j) for all $2 \leq j \leq k + 1$. The existence of a hamiltonian path in H_1 and a k -DPC in H_0 is straightforward.

For the remaining case that \bar{s} is a sink, we let $t_1 = \bar{s}$. Find a hamiltonian cycle $C = (t_2, \dots, z, t_3, \dots, u)$ in $H_1 \setminus (F_1 \cup F'')$, where $F'' = \{t_1\} \cup \{t_4, \dots, t_{k+1}\}$. The C exists since H_1 is $f + k - 1$ -fault hamiltonian and $f_1 + |F''| = f_1 + (k - 1) \leq f + k - 1$. We let $Q_1 = (t_2, \dots, z)$ and $Q_2 = (t_3, \dots, u)$ be two disjoint paths such that $C = (Q_1, Q_2)$. Find k -DPC $[s, \{\bar{z}, \bar{u}, \bar{t}_4, \dots, \bar{t}_{k+1}\} | H_0, \emptyset]$, and then merge the k -DPC and C with the edges (s, \bar{s}) , (\bar{z}, z) , (\bar{u}, u) , and (\bar{t}_j, t_j) for all $4 \leq j \leq k + 1$.

Case 2: $f_0 + f_2 = 0$ and $k_0 = 1$.

Let t_{k+1} be the sink in H_0 . We consider the case that \bar{s} is either a free vertex or a faulty vertex first. We find k -DPC $[v, \{t_1, t_2, \dots, t_k\} | H_1, F_1]$. We let the v - t_j path in the k -DPC be (v, z_j, \dots, t_j) for $1 \leq j \leq k$, and assume w.l.o.g. that $\bar{z}_j \neq t_{k+1}$ for every $2 \leq j \leq k$. We find k -DPC $[s, \{\bar{z}_2, \dots, \bar{z}_k, t_{k+1}\} | H_0, F']$, and then merge the two k -DPC's with the free bridge of s and (\bar{z}_j, z_j) for all $2 \leq j \leq k$. A $k+1$ -DPC of $H_0 \oplus H_1$ can be obtained by removing all the edges (v, z_j) , $2 \leq j \leq k$. It is easy to observe the existence of two k -DPC's.

When $\bar{s} \in T$, we let $t_1 = \bar{s}$ and assume w.l.o.g. that $\bar{t}_j \neq t_{k+1}$ for all $3 \leq j \leq k$. Find a hamiltonian cycle $C = (t_2, z, \dots, u)$ in $H_1 \setminus (F_1 \cup F'')$, where $F'' = \{t_1\} \cup \{t_3, \dots, t_k\}$. We assume w.l.o.g. that $\bar{z} \neq t_{k+1}$. Find k -DPC $[s, \{\bar{z}, \bar{t}_3, \dots, \bar{t}_k, t_{k+1}\} | H_0, \emptyset]$, and then merge the k -DPC and C with the edges (s, \bar{s}) , (\bar{z}, z) , and (\bar{t}_j, t_j) for all $3 \leq j \leq k$.

Case 3: $f_0 + f_2 = 1$ and $k_0 = 0$.

We assume w.l.o.g. that (t_1, \bar{t}_1) is the free bridge of t_1 . Let us first consider the case that either (s, \bar{s}) is a free bridge of s or at least one of (s, \bar{s}) and \bar{s} are faulty elements. We find k -DPC $[v, \{t_2, \dots, t_{k+1}\} | H_1, F_1 \cup \{t_1\}]$. The k -DPC exists since $f_1 + 1 = f_1 + (f_0 + f_2) \leq f$. We let the v - t_j path in the k -DPC be (v, z_j, \dots, t_j) , $2 \leq j \leq k+1$. We assume w.l.o.g. that (z_j, \bar{z}_j) is the free bridge of z_j for all $2 \leq j \leq k$. Find k -DPC $[s, \{\bar{t}_1, \bar{z}_2, \dots, \bar{z}_k\} | H_0, F_0 \cup F']$. The existence of the k -DPC is due to the fact that $|F'| = 1$ if and only if $f_1 + f_2 \geq 1$ (equivalently, $f_0 \leq f - 1$). Finally, we merge the two k -DPC's with the free bridge of s and the edges (\bar{t}_1, t_1) , and (\bar{z}_j, z_j) for all $2 \leq j \leq k$.

Now, (s, \bar{s}) is a fault-free edge and \bar{s} is a sink. We let $t_1 = \bar{s}$. Find k -DPC $[y, \{t_2, \dots, t_{k+1}\} | H_1, F_1 \cup \{t_1\}]$, and let the y - t_j path in the k -DPC be (y, z_j, \dots, t_j) , $2 \leq j \leq k+1$. We assume w.l.o.g. that (z_j, \bar{z}_j) is the free bridge of z_j for every $2 \leq j \leq k$. We find k -DPC $[s, \{\bar{z}_2, \dots, \bar{z}_k\} \cup \{x\} | H_0, F_0]$. Finally, we merge the two k -DPC's in a very similar way to the previous case.

2.2 When C1: $k_1 = 1$, $\bar{s} \in T$, $f_0 = f$

We let $\bar{s} = t_1$. Find k -DPC $[s, \{t_2, \dots, t_{k+1}\} | H_0, F_0]$. Since $f + k \leq \delta - 1$, there exists j , $2 \leq j \leq k+1$, such that the s - t_j path P_j in the k -DPC passes through at least two vertices z and u adjacent to s satisfying $(s, z), (s, u) \notin F$. Let $P_j = (Q_1, Q_2)$, where $Q_1 = (s, z, \dots, x)$, $Q_2 = (u, \dots, t_j)$. That is, $P_j = (s, z, \dots, x, u, \dots, t_j)$. We redefine P_j to be $P_j = (s, Q_2) = (s, u, \dots, t_j)$, and construct s - t_1 path in a way that $P_1 = (Q_1, \bar{x}, H[\bar{x}, t_1 | H_1, \emptyset])$. All the s - t_j paths form a $k+1$ -DPC in $H_0 \oplus H_1$.

2.3 When $k_1 = 0$

If (s, \bar{s}) is not the free bridge of s , we let (s, x, y) be the free bridge of s . First, let us consider the case that either $f_1 + f_2 = 0$ or $f_1 + f_2 = 1$ and (s, \bar{s}) is not the free bridge of s . Obviously, when $f_1 + f_2 = 1$ and (s, \bar{s}) is not the free bridge of s , $\bar{s} \in F$ or $(s, \bar{s}) \in F$. Thus, in this case, for every vertex v in H_0 different from s , (v, \bar{v}) is fault-free. We let

$$F' = \begin{cases} \emptyset, & \text{if } f_1 + f_2 = 0; \\ \{x\}, & \text{if } f_1 + f_2 = 1 \text{ and } (s, \bar{s}) \text{ is not the free bridge of } s. \end{cases}$$

We first find k -DPC $[s, \{t_1, t_2, \dots, t_k\} | H_0, F_0 \cup F']$. Here, sink t_{k+1} is regarded as a free vertex. The k -DPC exists since $f_0 + |F'| \leq f$. Some s - t_j path P_j in the k -DPC passes through t_{k+1} , and let $P_j = (s, Q_1, t_{k+1}, z, Q_2, t_j)$. Define s - t_{k+1} path to be (s, Q_1, t_{k+1}) . We denote by B_s the free bridge of s , and let v be the endvertex of B_s in H_1 . Redefine P_j to be $P_j = (B_s, H[v, \bar{z} | H_1, F_1], z, Q_2, t_j)$. All of the s - t_j paths form a $k+1$ -DPC in $H_0 \oplus H_1$.

Now, we consider the case that either $f_1 + f_2 \geq 2$ or $f_1 + f_2 = 1$ and (s, \bar{s}) is the free bridge of s . If there is a sink t_j such that (t_j, \bar{t}_j) is the free bridge of t_j , then we let t_1 be such a sink. If there is no such a sink, we let t_1 be an arbitrary sink and let (t_1, z, \bar{z}) be the free bridge of t_1 . We let $F' = F'_s \cup F'_{t_1}$, where

$$F'_s = \begin{cases} \emptyset, & \text{if } (s, \bar{s}) \text{ is the free bridge of } s; \\ \{x\}, & \text{if } (s, \bar{s}) \text{ is not the free bridge of } s, \text{ and} \end{cases}$$

$$F'_{t_1} = \begin{cases} \{t_1\}, & \text{if } (t_1, \bar{t}_1) \text{ is the free bridge of } t_1; \\ \{t_1, z\}, & \text{if } (t_1, \bar{t}_1) \text{ is not the free bridge of } t_1. \end{cases}$$

We find k -DPC $[s, \{t_2, \dots, t_{k+1}\} | H_0, F_0 \cup F']$, and then find $H[v, w | H_1, F_1]$, where w is the endvertex of the free bridge of t_1 in H_1 . The existence of such a k -DPC is due to the following lemma. The k -DPC and an s - t_1 path $P_1 = (s, B_s, H[v, w | H_1, F_1], B_1, t_1)$ form a $k+1$ -DPC, where B_1 is the free bridge of t_1 .

Lemma 5. *If either $f_1 + f_2 \geq 2$ or $f_1 + f_2 = 1$ and (s, \bar{s}) is a free bridge of s , then $f_0 + f' \leq f$, where $f' = |F'|$.*

Proof. Observe that if (s, \bar{s}) is not the free bridge of s , then at least one of \bar{s} and (s, \bar{s}) are faulty elements. Similarly, if (t_j, \bar{t}_j) is not the free bridge of t_j , then $\bar{t}_j \in F$ or $(t_j, \bar{t}_j) \in F$. When (t_1, \bar{t}_1) is not the free bridge of t_1 , $f_1 + f_2 \geq k+1$, and thus $f_0 \leq f - k - 1$ and $f_0 + f' \leq f_0 + 3 \leq (f - k - 1) + 3 \leq f$. Recall that $k \geq 2$. Now, let us consider the case that (t_1, \bar{t}_1) is the free bridge of t_1 . If (s, \bar{s}) is the free bridge of s , $f_0 + f' = f_0 + 1 \leq f$ since $f_1 + f_2 \geq 1$. If (s, \bar{s}) is not the free bridge of s , $f_1 + f_2 \geq 2$ by assumption, and thus $f_0 + f' = f_0 + 2 \leq f$. \square

3 Application

A graph G is called *fully one-to-many disjoint path coverable* if for any $f \geq 0$ and $k \geq 2$ such that $f + k \leq \delta(G) - 1$, G is f -fault one-to-many k -disjoint path coverable. A graph G is called *almost fully one-to-many disjoint path coverable* if for any $f \geq 0$ and $k \geq 3$ such that $f + k \leq \delta(G) - 1$, G is f -fault one-to-many k -disjoint path coverable. Note that almost fully one-to-many disjoint path coverable graph G which is $\delta(G) - 3$ -fault hamiltonian-connected is fully one-to-many disjoint path coverable. The following is immediate from Theorem 1.

Corollary 1. *Let H_i be a graph with n vertices satisfying the following two conditions, where $\delta_i = \delta(H_i)$, $i = 0, 1$.*

- (a) H_i is fully one-to-many disjoint path coverable.
 (b) H_i is $\delta_i - 2$ -fault hamiltonian.

Then, $H_0 \oplus H_1$ is almost fully one-to-many disjoint path coverable.

3.1 Recursive Circulants $G(2^m, 4)$

$G(2^m, 4)$ is an m -regular graph with 2^m vertices. According to the recursive structure of recursive circulants[6], we can observe that $G(2^m, 4)$ is isomorphic to $G(2^{m-2}, 4) \times K_2 \oplus_M G(2^{m-2}, 4) \times K_2$ for some permutation M . Obviously, $G(2^{m-2}, 4) \times K_2$ is isomorphic to $G(2^{m-2}, 4) \oplus_{M'} G(2^{m-2}, 4)$ for some M' . First we consider fault-hamiltonicity of $G(2^m, 4)$ and $G(2^m, 4) \times K_2$, and then we show that $G(2^m, 4)$, $m \geq 3$, is fully one-to-many disjoint path coverable. Lemma 7 implies that $G(2^{m-1}, 4) \times K_2$ of degree m , $m \geq 4$, is $m - 3$ -fault hamiltonian-connected and $m - 2$ -fault hamiltonian. The proof of Lemma 7 is omitted in this paper due to space limit.

Lemma 6. [7] $G(2^m, 4)$, $m \geq 3$, is $m - 3$ -fault hamiltonian-connected and $m - 2$ -fault hamiltonian.

Lemma 7. If a graph G is $\delta(G) - 3$ -fault hamiltonian-connected and $\delta(G) - 2$ -fault hamiltonian for $\delta(G) \geq 3$, then $G \times K_2$ is $\delta(G \times K_2) - 3$ -fault hamiltonian-connected and $\delta(G \times K_2) - 2$ -fault hamiltonian.

Theorem 2. $G(2^m, 4)$, $m \geq 3$, is fully one-to-many disjoint path coverable.

Proof. The proof is by induction on m . For $m = 3$, the theorem holds true by Lemma 6. For $m = 4$, by Lemma 6, it is sufficient to show that $G(2^4, 4)$ is 0-fault one-to-many 3-disjoint path coverable. The proof is mainly by case analysis, and omitted here. For $m \geq 5$, by Corollary 1 and Lemma 7, $G(2^{m-2}, 4) \times K_2$ is fully one-to-many disjoint path coverable, and thus, by Corollary 1 and Lemma 6, $G(2^m, 4)$ is fully one-to-many disjoint path coverable. \square

3.2 Twisted Cube TQ_m , Crossed Cube CQ_m

Originally, twisted cube TQ_m is defined for odd m . We let $TQ_m = TQ_{m-1} \times K_2$ for even m . Then, TQ_m is isomorphic to $TQ_{m-1} \oplus_M TQ_{m-1}$ for some M . Also, crossed cube CQ_m is isomorphic to $CQ_{m-1} \oplus_{M'} CQ_{m-1}$ for some M' . Both TQ_m and CQ_m are m -regular graphs with 2^m vertices. The fault-hamiltonicity of them were studied in the literature.

Lemma 8. (a) TQ_m , $m \geq 3$, is $m - 3$ -fault hamiltonian-connected and $m - 2$ -fault hamiltonian[4]. (b) CQ_m , $m \geq 3$, is $m - 3$ -fault hamiltonian-connected and $m - 2$ -fault hamiltonian[3].

By Corollary 1 and Lemma 7 and 8, we have the following.

Theorem 3. TQ_m and CQ_m , $m \geq 3$, are fully one-to-many disjoint path coverable.

3.3 One-to-One Disjoint Path Covers

Similar to the definition of a one-to-many disjoint path cover, we can define a one-to-one disjoint path cover in a graph with faulty elements. In a straightforward manner, when $f + k \leq \delta(G) - 1$, we can construct f -fault one-to-one k -DPC of a graph G joining s and t using f -fault one-to-many k -DPC joining s and $\{t_1, t_2, \dots, t_k\}$, where $t_1 = t$ and for every $2 \leq j \leq k$, t_j is a fault-free vertex adjacent to t such that (t_j, t) is also fault-free.

Proposition 2. *Let G be a graph, and let $f \geq 0$. (a) G is f -fault one-to-one 1-disjoint path coverable if and only if G is f -fault hamiltonian-connected. (b) G is f -fault one-to-one 2-disjoint path coverable if and only if G is f -fault hamiltonian. (c) Let $k \geq 1$, and $f + k \leq \delta(G) - 1$. If G is f -fault one-to-many k -disjoint path coverable, then G is f -fault one-to-one k -disjoint path coverable.*

Theorem 4. *For $m \geq 3$, $G(2^m, 4)$, TQ_m , and CQ_m are f -fault one-to-one k -disjoint path coverable for any $f \geq 0$ and $k \geq 2$ such that $f + k \leq m - 1$.*

It was shown that fault-free $G(2^m, 4)$, $m \geq 3$, is (0-fault) one-to-one k -disjoint path coverable for any $1 \leq k \leq m$ [5]. The construction of f -fault one-to-one k -DPC's of $G(2^m, 4)$, TQ_m , and CQ_m satisfying $f + k = m$ remains open.

References

1. D.-R. Duh and G.-H. Chen, "On the Rabin number problem," *Networks* **30**(3), pp. 219-230, 1997.
2. F. Harary and M. Lewinter, "The starlike trees which span a hypercube," *Computers & Mathematics with Applications* **15**(4), pp. 299-302, 1988.
3. W.T. Huang, M.Y. Lin, J.M. Tan, and L.H. Hsu, "Fault-tolerant ring embedding in faulty crossed cubes," in *Proc. SCI'2000*, pp. 97-102, 2000.
4. W.T. Huang, J.M. Tan, C.N. Huang, L.H. Hsu, "Fault-tolerant hamiltonicity of twisted cubes," *J. Parallel Distrib. Comput.* **62**, pp. 591-604, 2002.
5. J.-H. Park, "One-to-one disjoint path covers in recursive circulants," *Journal of KISS* **30**(12), pp. 691-698, 2003 (in Korean).
6. J.-H. Park and K.Y. Chwa, "Recursive circulants and their embeddings among hypercubes," *Theoretical Computer Science* **244**, pp. 35-62, 2000.
7. C.-H. Tsai, J.M. Tan, Y.-C. Chuang, and L.-H. Hsu, "Fault-free cycles and links in faulty recursive circulant graphs," in *Proc. of Workshop on Algorithms and Theory of Computation ICS2000*, pp. 74-77, 2000.

Fault-Tolerant Meshes with Constant Degree

Toshinori Yamada

Department of Information and Computer Sciences, Saitama University
255 Shimo-Okubo, Sakura-ku, Saitama-shi, Saitama 358-8570, Japan
yamada@pd.ics.saitama-u.ac.jp

Abstract. This paper proves that for every positive integers n, k and any positive constant ε , we can explicitly construct a graph G with $n + O(k^{1+\varepsilon})$ vertices and a constant degree such that even after removing any k vertices from G , the remaining graph still contains an n -vertex dipath. This paper also proves that for every positive integers n, k and any positive constant ε , we can explicitly construct a graph H with $n + O(k^{2+\varepsilon})$ vertices and a constant degree such that even after removing any k vertices from H , the remaining graph still contains an n -vertex 2-dimensional square mesh.

1 Introduction

We consider the following problem motivated by the design of fault-tolerant multiprocessor systems. Let G be an undirected or directed graph. If G is undirected then let $V(G)$ and $E(G)$ denote the vertex set and edge set of G , respectively, and (u, v) denote the edge connecting u and v in G . If G is directed then let $V(G)$ and $A(G)$ denote the vertex set and arc set of G , respectively, and $\langle u, v \rangle$ denote the arc from u to v in G . $\Delta(G)$ is the maximum degree of a vertex in G . For any $S \subseteq V(G)$, $G - S$ is the graph obtained from G by deleting the vertices of S together with the edges incident with the vertices in S . Let k be a positive integer. A graph G is called a k -FT (k -fault-tolerant) graph for a graph H if $G - F$ contains H as a subgraph for every $F \subseteq V(G)$ with $|F| \leq k$. If G is a DAG (directed acyclic graph), which implies that H is also a DAG, then G is called a k -FT DAG for H . Our problem is to construct a k -FT graph G for a given graph H such that $|V(G)|$ and $\Delta(G)$ are as small as possible.

A large amount of research has been devoted to constructing k -FT graphs for an n -vertex (undirected) path P_n [1, 4, 5, 8–10, 13–16], k -FT DAGs for an n -vertex dipath (directed path) \hat{P}_n [4, 6, 14–16], and k -FT graphs for a 2-dimensional m -sided mesh $M_2(m)$ [2–4, 7, 11, 14–16].

Among others, Bruck, Cypher, and Ho [4] show a k -FT DAG for \hat{P}_n with $n + O(k^2)$ vertices and maximum degree of 4, and a k -FT graph for $M_2(m)$ with $m^2 + O(k^3)$ vertices and a constant maximum degree. Zhang [15, 16] shows a k -FT graph for P_n with $n + O(k \log^2 k)$ vertices and maximum degree of $O(1)$, a k -FT DAG for \hat{P}_n with $n + O(k \log k)$ vertices and maximum degree of $O(\log k)$, and a k -FT graph for $M_2(m)$ with $m^2 + O(k^2 / \log k)$ vertices and $O(\log^3 k)$ maximum degree. Zhang [15, 16] also raised the following three open questions:

- (Q1) Is it possible to construct a k -FT graph for an n -vertex path P_n with $n + O(k)$ vertices and a constant maximum degree?
- (Q2) Is it possible to construct a k -FT DAG for an n -vertex dipath \hat{P}_n with $n + O(k \log k)$ vertices and a constant maximum degree?
- (Q3) Is it possible to construct a k -FT graph for a 2-dimensional m -sided mesh $M_2(m)$ with $m^2 + O(k^2)$ vertices and a constant maximum degree?

Yamada and Ueno [14] settle (Q1) by showing an explicit construction of k -FT graph for P_n with $n + O(k)$ vertices and maximum degree of 3.

This paper considers (Q2) and (Q3), and shows the explicit constructions of a k -FT DAG for \hat{P}_n with $n + O(k^{1+\varepsilon})$ vertices and a constant maximum degree, and a k -FT graph for $M_2(m)$ with $m^2 + O(k^{2+\varepsilon})$ vertices and a constant maximum degree for any constant $\varepsilon > 0$. Our construction of k -FT graphs for $M_2(m)$ is based on the method proposed by [15, 16] combined with our k -FT DAG for \hat{P}_n .

2 Fault-Tolerant Directed Linear Arrays

For any positive integer n , let $[n] = \{0, 1, \dots, n-1\}$. The n -vertex dipath \hat{P}_n is the digraph defined as follows:

$$V(\hat{P}_n) = [n]; \quad A(\hat{P}_n) = \{\langle u, v \rangle : u, v \in [n], v = u + 1\}.$$

For any positive integers N , t , and d , we define $\hat{D}_{N,t,d}$ as the following DAG:

$$V(\hat{D}_{N,t,d}) = [N]; \quad A(\hat{D}_{N,t,d}) = \{\langle u, v \rangle : v = u + t^x \text{ for } x \in [d+1]\}.$$

Then, we can prove the following lemma.

Lemma 1. *Let N, t, d, k , and n be positive integers. If $k < t^d$ and*

$$N \geq \left\lceil \frac{n + dk(t-1) - 1}{t^d} + 1 \right\rceil \times t^d$$

then $\hat{D}_{N,t,d}$ is a k -FT DAG for \hat{P}_n .

Proof. Assume without loss of generality that

$$N = \left\lceil \frac{n + dk(t-1) - 1}{t^d} + 1 \right\rceil \times t^d.$$

Notice that N/t^x is a positive integer for any $x \in [d+1]$. Fix any $F \subseteq V(\hat{D}_{N,t,d})$ with $|F| \leq k$. A vertex in F is said to be faulty. We prove that $\hat{D}_{N,t,d} - F$ contains \hat{P}_n as a subdigraph.

For any $x \in [d+1]$ and $y \in [t^x]$, define that $U(x, y) = \{y + jt^x : j \in [N/t^x]\}$. First, we prove the following claim.

Claim 1. There exists a sequence of integers, y_0, y_1, \dots, y_d , such that $y_0 = 0$, $(y_{x+1} - y_x)/t^x \in [t]$ for any $x \in [d]$, and $U(x, y_x)$ has at most k/t^x faulty vertices for any $x \in [d+1]$.

Proof of Claim 1. We prove the claim by constructing the sequence inductively.

$U(0, y_0) = V(\hat{D}_{n,t,d})$ has at most $k/t^0 = k$ faulty vertices by assumption.

Let $\delta \in [d]$, and assume that there exists a sequence of $y_0, y_1, \dots, y_\delta$ such that $y_0 = 0$, $(y_{x+1} - y_x)/t^x \in [t]$ for any $x \in [\delta]$, and $U(x, y_x)$ has at most k/t^x faulty vertices for any $x \in [\delta+1]$. Since $U(\delta, y_\delta)$ has at most k/t^δ faulty vertices and $(U(\delta+1, y_\delta), U(\delta+1, y_\delta + t^\delta), \dots, U(\delta+1, y_\delta + (t-1)t^\delta))$ is a partition of $U(\delta, y_\delta)$, there exists some $i_{\delta+1} \in [t]$ such that $U(\delta+1, y_\delta + i_{\delta+1}t^\delta)$ has at most $(k/t^\delta)/\delta = k/t^{\delta+1}$ faulty vertices. Hence, the claim holds by putting $y_{\delta+1} = y_\delta + i_{\delta+1}t^\delta$. \square

Let y_0, y_1, \dots, y_d be a sequence of integers satisfying the condition in Claim 1. For any $x \in [d+1]$, let k_x denote the number of faulty vertices in $U(x, y_x)$. By Claim 1, we obtain that $k_x \leq k/t^x$. Since $k_d \leq k/t^d < 1$, we obtain $k_d = 0$. That is, $U(d, y_d)$ has no faulty vertices.

We inductively construct $d+1$ fault-free dipaths, denoted by $\hat{L}(d), \hat{L}(d-1), \dots, \hat{L}(0)$, from y_d to $y_d + (N - t^d)$ as follows:

(i) Define that $\hat{L}(d) = \langle y_d, y_d + t^d, \dots, y_d + (N - t^d) \rangle$. Then $\hat{L}(d)$ is a fault-free dipath from y_d to $y_d + (N - t^d)$ since every vertex of $U(d, y_d) = \{y_d + jt^d : j \in [N/t^d]\}$ is fault-free,.

(ii) Assume that a fault-free dipath $\hat{L}(x+1)$ from y_d to $y_d + (N - t^d)$ is already constructed. Then, we define $\hat{L}(x)$ as the dipath obtained from $\hat{L}(x+1)$ by replacing each arc $\langle v, v + t^{x+1} \rangle \in A(\hat{L}(x+1))$ with a dipath $\hat{L}' = \langle v, v + t^x, \dots, v + (t-1)t^x, v + t^{x+1} \rangle$ if every vertex of \hat{L}' is fault-free. By the definition of $\hat{L}(x)$, $\hat{L}(x)$ is a fault-free path from y_d to $y_d + (N - t^d)$.

We prove that at least n vertices. Let r_x be the number of arcs $\langle v, v + t^x \rangle \in A(\hat{L}(0))$. Since $r_{x+1} \leq k_x - k_{x+1}$ for any $x \in [d]$, we obtain

$$\begin{aligned} & |V(\hat{L}(0))| \\ &= (N - t^d + 1) - \sum_{x=1}^d r_x(t^x - 1) \geq n + dk(t-1) - \sum_{x=1}^d (k_{x-1} - k_x)(t^x - 1) \\ &= n + dk(t-1) - \sum_{x=0}^{d-1} k_x(t^{x+1} - t^x) \geq n + dk(t-1) - \sum_{x=0}^{d-1} (k/t^x)(t^{x+1} - t^x) \\ &= n + dk(t-1) - \sum_{x=0}^{d-1} k(t-1) = n + dk(t-1) - dk(t-1) = n. \end{aligned}$$

Therefore, $\hat{L}(0)$ is a dipath with at least n vertices. Since every vertex of $\hat{L}(0)$ is fault-free, $\hat{D}_{N,t,d} - F$ contains \hat{P}_n as a subdigraph, and hence we conclude that $\hat{D}_{N,t,d}$ is a k -FT DAG for \hat{P}_n . \square

Theorem 1. For any positive integers n, k and any constant $\varepsilon > 0$, we can construct a k -FT DAG for \hat{P}_n with $n + O(k^{1+\varepsilon})$ vertices and a constant maximum vertex degree.

Proof. Fix $d = \lceil 1/\varepsilon \rceil$, and set t and N as integers satisfying that $(t-1)^d \leq k < t^d$ and

$$N = \left\lceil \frac{n + dk(t-1) - 1}{t^d} + 1 \right\rceil \times t^d.$$

By Lemma 1, $\hat{D}_{N,t,d}$ is a k -FT DAG for \hat{P}_n . By the definition of $\hat{D}_{N,t,d}$, the maximum degree of a vertex in $\hat{D}_{N,t,d}$ are at most $2(d+1)$, which is a constant. Since $t \leq k^{1/d} + 1$, we obtain

$$N \leq n + dk^{1+(1/d)} + 2(k^{1/d} + 1)^d = n + O(k^{1+\varepsilon}),$$

which completes the proof of the theorem. \square

3 Fault-Tolerant Meshes

For any positive integers r and c , the 2-dimensional $r \times c$ mesh, denoted by $M(r, c)$, is the graph defined as follows:

$$\begin{aligned} V(M(r, c)) &= [r] \times [c]; \\ E(M(r, c)) &= \{([i, j], [i', j']) : |i - i'| + |j - j'| = 1\}. \end{aligned}$$

$M(m, m)$ is denoted by $M_2(m)$, which is called the 2-dimensional m -sided mesh.

3.1 Basic Construction

Let N, t, d, r , and c be any positive integers. $G_{N,t,d}^{r,c}$ is the graph defined as follows:

$$V(G_{N,t,d}^{r,c}) = [N]; \quad E(G_{N,t,d}^{r,c}) = E_{\text{hor}}(G_{N,t,d}^{r,c}) \cup E_{\text{ver}}(G_{N,t,d}^{r,c}),$$

where

$$\begin{aligned} E_{\text{hor}}(G_{N,t,d}^{r,c}) &= \{(u, v) : |v - u| = t^x \text{ for } x \in [d+1]\} \text{ and} \\ E_{\text{ver}}(G_{N,t,d}^{r,c}) &= \{(u, v) : |v - u| - c \in [N - rc + 1]\}. \end{aligned}$$

Lemma 2. *If $k < t^d$ and*

$$N \geq \left\lceil \frac{rc + dk(t-1) - 1}{t^d} + 1 \right\rceil \times t^d$$

then $G_{N,t,d}^{r,c}$ is a k -FT graph for $M(r, c)$.

Proof. In order to prove the lemma, it suffices to show the following claim:

Claim 2. For any $F \subseteq V(G_{N,t,d}^{r,c})$ with $|F| \leq k$, there exist rc vertices $v_0, v_1, \dots, v_{rc-1} \in V(G_{N,t,d}^{r,c}) - F$ satisfying the following three conditions:

1. $v_0 < v_1 < \dots < v_{rc-1}$;
2. $(v_x, v_{x+1}) \in E_{\text{hor}}(G_{N,t,d}^{r,c})$ for every $x \in [rc-1]$; and
3. $(v_x, v_{x+c}) \in E_{\text{ver}}(G_{N,t,d}^{r,c})$ for every $x \in [rc-c]$.

Proof of Claim 2. By Lemma 1, there exist rc vertices $v_0, v_1, \dots, v_{rc-1} \in V(G_{N,t,d}^{r,c}) - F$ satisfying conditions 1 and 2. Since

$$|V(G_{N,t,d}^{r,c}) - \{v_x : x \in [rc]\}| = N - rc,$$

we obtain that $c \leq v_{x+c} - v_x \leq c + N - rc$ for every $x \in [rc - c]$, and so $(v_{x+c}, v_x) \in E_{\text{ver}}(G_{N,t,d}^{r,c})$. \square

By Claim 2, $G_{N,t,d}^{r,c} - F$ contains $M(r, c)$ as a subgraph for any $F \subseteq V(G_{N,t,d}^{r,c})$ with $|F| \leq k$, and hence $G_{N,t,d}^{r,c}$ is a k -FT graph for $M(r, c)$. \square

By Lemma 2, we have the following theorem.

Theorem 2. *For any positive integers r, c , and k , and any constant $\varepsilon > 0$, we can construct a k -FT graph for $M(r, c)$ with $rc + O(k^{1+\varepsilon})$ vertices and maximum vertex degree of $O(k^{1+\varepsilon})$.*

3.2 Tower Graphs

For any positive integer h , the tower graph $Q(h)$ of height h is the graph defined as follows:

$$\begin{aligned} V(Q(h)) &= \{[l, p] : l \in [h+1], p \in [2^{h-l}]\}; \\ E(Q(h)) &= E_{\text{tr}}(Q(h)) \cup E_{\text{lp}}(Q(h)), \end{aligned}$$

where

$$\begin{aligned} E_{\text{tr}}(Q(h)) &= \{([l, p], [l+1, q]) : q = \lfloor p/2 \rfloor\} \text{ and} \\ E_{\text{lp}}(Q(h)) &= \{([l, p], [l, q]) : q = (p \pm 1) \bmod 2^{h-l}\}. \end{aligned}$$

l and p are called the level and position of a vertex $[l, p]$, respectively. An edge in $E_{\text{tr}}(Q(h)) \setminus E_{\text{lp}}(Q(h))$ is called a tree[loop] edge. $[h, 0]$ and $[0, p]$ are called a root and leaf of $Q(h)$, respectively. It is easy to see that $|V(Q(h))| = 2^{h+1} - 1$ and $\Delta(Q(h)) \leq 5$. Fig.1 shows $Q(3)$.

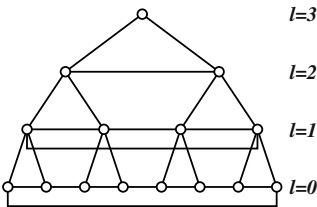


Fig. 1. Tower Graph $Q(3)$

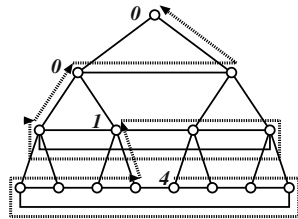


Fig. 2. Hamilton Path $H(3, 4)$ in $Q(3)$

For any positive integer h , $p \in [2^h]$, and $l \in [h+1]$, $w_l(h, p)$ is defined as follows: $w_0(h, p) = p$; $w_{l+1}(h, p) = \lfloor (w_l(h, p) - 1)/2 \rfloor$ for any $l \in [h]$. For any

$p \in [2^h]$, let $H(h, p)$ denote the Hamilton path of $Q(h)$ from $[0, p]$ to $[h, 0]$ defined as follows: Iterate the following through $l = 0$ to $h - 1$:

- Traverse loop edges in the order of $[l, w_l(h, p)], [l, w_l(h, p) + 1], \dots, [l, 2^{h-l} - 1], [l, 0], \dots, [l, w_l(h, p) - 1]$;
- Go up from $[l, w_l(h, p) - 1]$ to $[l + 1, w_{l+1}(h, p)]$ using a tree edge.

Fig.2 shows $H(3, 4)$. It is easy to see that $w_0(3, 4) = 4, w_1(3, 4) = 1, w_2(3, 4) = 0$, and $w_3(3, 4) = 0$. See Fig.2.

For any positive integer $i \leq 2^{h+1} - 1$, let $s(h, p, i)$ denote the i -th vertex from $[0, p]$ in $H(h, p)$. The following lemma is proved in [15, 16].

Lemma 3. [15, 16] *For any $p, q \in [2^h]$, $p < q$, and any positive integer $i \leq 2^{h+1} - 1$, $s(h, p, i)$ and $s(h, q, i)$ are vertices on the same level l , and*

$$\{\pi(h, q, i) - \pi(h, p, i)\} \bmod 2^{h-l} - \left\lfloor \frac{q-p}{2^l} \right\rfloor \in [2],$$

where $\pi(h, p, i)$ and $\pi(h, q, i)$ are the positions of $s(h, p, i)$ and $s(h, q, i)$, respectively.

3.3 Improved Construction

In this subsection, for any positive integers m, k , and d , we construct a graph $\mathcal{G}_{k,d}(m)$, which is shown to be a k -FT graph for $M_2(m)$ with $m^2 + O(d^2 k^{2+(2/d)})$ vertices and maximum degree of $O(d^3)$.

Set t as an integer with $(t-1)^d \leq k < t^d$, and $h = \lceil \log(dk(t-1) + 2t^d) \rceil$. For simplicity, we assume that $2^{h+2} - 2$ is a divisor of m . Let $r = m/(2^{h+2} - 2)$, $c = m$, and

$$N = \left\lceil \frac{rc + dk(t-1) - 1}{t^d} + 1 \right\rceil \times t^d.$$

Notice that $N - rc + 1 \leq dk(t-1) + 2t^d \leq 2^h$.

We define $\mathcal{Q}(h)$ as the graph obtained from two copies, $Q^0(h)$ and $Q^1(h)$, of $Q(h)$ by connecting a root of $Q^0(h)$ and that of $Q^1(h)$ by an edge. Fig.3 shows $\mathcal{Q}(3)$. For any $p, q \in [2^h]$, let $H^0(h, p)$ and $H^1(h, q)$ denote the Hamilton paths of $Q^0(h)$ and $Q^1(h)$ corresponding to $H(h, p)$ and $H(h, q)$ in $Q(h)$, respectively. $\mathcal{H}(h, p, q)$ is the Hamilton path of $\mathcal{Q}(h)$ obtained from $H^0(h, p)$ and $H^1(h, q)$ by connecting a root of $Q^0(h)$ and that of $Q^1(h)$ by an edge. Let $s'(h, p, q, i)$ denote the i -th vertex from $[0, p] \in V(Q^0(h))$ in $\mathcal{H}(h, p, q)$.

For any positive integers t, d , and l , define that

$$S_{t,d} = \{t^\alpha - t^\beta : 0 \leq \alpha \leq \beta \leq d\} \text{ and } \mathcal{S}_{t,d,l} = \left\{ \left\lfloor \frac{x}{2^l} \right\rfloor + b : x \in S_{t,d}, b \in [2] \right\}.$$

$\mathcal{G}_{k,d}(m)$ is the graph obtained from $G_{N,t,d}^{r,c}$ as follows:

(C1) Replace each vertex v of $G_{N,t,d}^{r,c}$ with a copy \mathcal{Q}_v of $\mathcal{Q}(h)$;

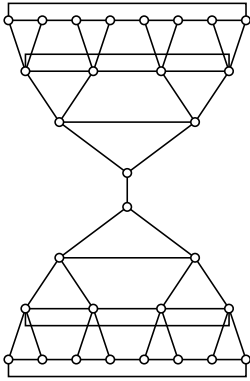


Fig. 3. Graph $\mathcal{Q}(3)$

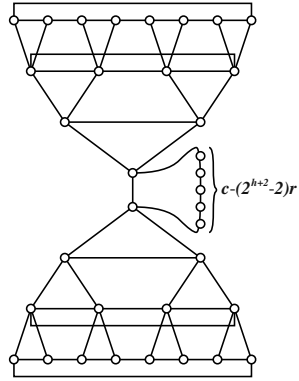


Fig. 4. Graph constructed in (C'2)

- (C2) If $(u, v) \in E_{\text{hor}}(G_{N,t,d}^{r,c})$ then connect each vertex $[l, p]$ in $Q_u^0[Q_u^1]$ with $[l, (p \pm \delta) \bmod 2^{h-l}]$ in $Q_v^0[Q_v^1]$ by an edge for every $\delta \in \mathcal{S}_{t,d,l}$, where $Q_v^0[Q_v^1]$ represents a copy of $Q(h)$ in \mathcal{Q}_v corresponding to $Q^0(h)[Q^0(h)]$ in $\mathcal{Q}(h)$;
- (C3) If $(u, v) \in E_{\text{ver}}(G_{N,t,d}^{r,c})$ and $u < v$ then connect a vertex $[0, p]$ in Q_u^1 with a vertex $[0, p]$ in Q_v^0 , where $p = (v - u) - c$.

It is easy to see the following two lemmas.

Lemma 4. $|V(\mathcal{G}_{k,d}(m))| = m^2 + O(d^2 k^{2+(2/d)})$.

Lemma 5. $\Delta(\mathcal{G}_{k,d}(m)) = O(d^3)$.

Lemma 6. $\mathcal{G}_{k,d}(m)$ is a k -FT graph for $M_2(m)$.

Proof. Fix any $F \subseteq V(\mathcal{G}_{k,d}(m))$ with $|F| \leq k$. Let $F' = \{v : \mathcal{Q}_v \text{ has a vertex in } F\}$. Since $|F'| \leq k$, there exist rc vertices $v_0, \dots, v_{rc-1} \in V(G_{N,t,d}^{r,c}) - F'$ satisfying the conditions in Claim 2. For any $x \in [r+1]$ and $y \in [c]$, let

$$z_{x,y} = \begin{cases} 0 & \text{if } x = 0 \text{ or } x = r, \\ (v_{y+xc} - v_{y+(x-1)c}) - c & \text{otherwise.} \end{cases}$$

We define a mapping $\phi : V(M_2(m)) \rightarrow V(\mathcal{G}_{k,d}(m)) - F$ as follows: For any $[m_1, m_2] \in V(M_2(m))$,

$$\phi([m_1, m_2]) = s'_{v_{y+xc}}(h, z_{x,y}, z_{x+1,y}, i),$$

where $s'_v(h, p, q, i)$ is the vertex in \mathcal{Q}_v corresponding to $s'(h, p, q, i)$ in $\mathcal{Q}(h)$, $x = \lfloor m_1 / (2^{h+2} - 2) \rfloor$, $i = m_1 \bmod (2^{h+2} - 2)$, and $y = m_2$.

Claim 3. For any $m_1 \in [m-1]$ and $m_2 \in [m]$, $\phi([m_1, m_2])$ is adjacent to $\phi([m_1 + 1, m_2])$.

Proof of Claim 3. Let $x = \lfloor m_1/(2^{h+2}-2) \rfloor$, $i = m_1 \bmod (2^{h+2}-2)$, and $y = m_2$. If $i \neq 2^{h+2} - 3$ then

$$\begin{aligned}\phi([m_1, m_2]) &= s'_{v_{y+xc}}(h, z_{x,y}, z_{x+1,y}, i) \\ \phi([m_1 + 1, m_2]) &= s'_{v_{y+xc}}(h, z_{x,y}, z_{x+1,y}, i + 1).\end{aligned}$$

Therefore, $\phi([m_1, m_2])$ is adjacent to $\phi([m_1 + 1, m_2])$.

If $i = 2^{h+2} - 3$ then

$$\begin{aligned}\phi([m_1, m_2]) &= s'_{v_{y+xc}}(h, z_{x,y}, z_{x+1,y}, 2^{h+2} - 3) \\ \phi([m_1 + 1, m_2]) &= s'_{v_{y+(x+1)c}}(h, z_{x+1,y}, z_{x+2,y}, 0).\end{aligned}$$

That is, $\phi([m_1, m_2])$ is a leaf $[0, z_{x+1,y}]$ in $Q_{v_{y+xc}}^1$ and $\phi([m_1 + 1, m_2])$ is a leaf $[0, z_{x+1,y}]$ in $Q_{v_{y+(x+1)c}}^0$. Since $z_{x+1,y} = (v_{y+(x+1)c} - v_{y+xc}) - c$, we conclude that $\phi([m_1, m_2])$ is adjacent to $\phi([m_1 + 1, m_2])$. \square

Claim 4. For any $m_1 \in [m]$ and $m_2 \in [m - 1]$, $\phi([m_1, m_2])$ is adjacent to $\phi([m_1, m_2 + 1])$.

Proof of Claim 4. Let $x = \lfloor m_1/(2^{h+2}-2) \rfloor$, $i = m_1 \bmod (2^{h+2}-2)$, and $y = m_2$. Then,

$$\begin{aligned}\phi([m_1, m_2]) &= s'_{v_{y+xc}}(h, z_{x,y}, z_{x+1,y}, i) \text{ and} \\ \phi([m_1, m_2 + 1]) &= s'_{v_{y+xc+1}}(h, z_{x,y+1}, z_{x+1,y+1}, i).\end{aligned}$$

Assume that $i \leq 2^{h+1} - 2$. Then,

$$\phi([m_1, m_2]) = s_{v_{y+xc}}^0(h, z_{x,y}, i) \text{ and } \phi([m_1, m_2 + 1]) = s_{v_{y+xc+1}}^0(h, z_{x,y+1}, i).$$

We consider two distinguished cases, (1) $x = 0$ and (2) $x \neq 0$.

(1) If $x = 0$ then

$$\phi([m_1, m_2]) = s_{v_y}^0(h, 0, i) \text{ and } \phi([m_1, m_2 + 1]) = s_{v_{y+1}}^0(h, 0, i).$$

That is, $\phi([m_1, m_2])$ and $\phi([m_1, m_2 + 1])$ are vertices of $Q_{v_y}^0$ and $Q_{v_{y+1}}^0$ corresponding to the same vertex of $Q(h)$, respectively. Since $(v_y, v_{y+1}) \in E_{\text{hor}}(G_{N,t,d}^{r,c})$ and $0 \in S_{t,d}$, we conclude that $\phi([m_1, m_2])$ is adjacent to $\phi([m_1, m_2 + 1])$.

(2) Assume that $x \neq 0$. Then, $z_{x,y} = (v_{y+xc} - v_{y+(x-1)c}) - c$ and $z_{x,y+1} = (v_{y+xc+1} - v_{y+(x-1)c+1}) - c$. Since $(v_{y+(x-1)c}, v_{y+(x-1)c+1}) \in E_{\text{hor}}(G_{N,t,d}^{r,c})$ and $(v_{y+xc}, v_{y+xc+1}) \in E_{\text{hor}}(G_{N,t,d}^{r,c})$, we obtain $v_{y+(x-1)c+1} - v_{y+(x-1)c} = t^\alpha$ and $v_{y+xc+1} - v_{y+xc} = t^\beta$ for some $\alpha, \beta \in [d + 1]$. Hence,

$$z_{x,y+1} - z_{x,y} = (v_{y+xc+1} - v_{y+xc}) - (v_{y+(x-1)c+1} - v_{y+(x-1)c}) = t^\beta - t^\alpha.$$

Assume without loss of generality that $\beta \geq \alpha$. Then, by Lemma 2, $s_{v_{y+xc}}^0(h, z_{x,y}, i)$ and $s_{v_{y+xc+1}}^0(h, z_{x,y+1}, i)$ are the same position l in $Q_{v_{y+xc}}^0$ and $Q_{v_{y+xc+1}}^0$, respectively, and

$$\begin{aligned}(\pi_{v_{y+xc+1}}^0(h, z_{x,y+1}, i) - \pi_{v_{y+xc}}^0(h, z_{x,y}, i)) \bmod 2^{h-l} \\ = \left\lfloor \frac{z_{x,y+1} - z_{x,y}}{2^l} \right\rfloor + b = \left\lfloor \frac{t^\beta - t^\alpha}{2^l} \right\rfloor + b,\end{aligned}$$

that is

$$\pi_{v_{y+xc+1}}^0(h, z_{x,y+1}, i) = \left(\pi_{v_{y+xc}}^0(h, z_{x,y}, i) + \left\lfloor \frac{t^\beta - t^\alpha}{2^l} \right\rfloor + b \right) \bmod 2^{h-l}$$

for some $b \in [2]$, where $\pi_{v_{y+xc}}^0(h, z_{x,y}, i)$ and $\pi_{v_{y+xc+1}}^0(h, z_{x,y+1}, i)$ are the positions of $s_{v_{y+xc}}^0(h, z_{x,y}, i)$ and $(s_{v_{y+xc+1}}^0(h, z_{x,y+1}, i))$, respectively. Since $t^\beta - t^\alpha \in S_{t,d}$, we conclude that $\phi([m_1, m_2])$ is adjacent to $\phi([m_1, m_2 + 1])$.

By a similar argument, we can also prove the claim when $2^{h+1} - 1 \leq i \leq 2^{h+2} - 2$. \square

By Claims 3 and 4, $\mathcal{G}_{k,d}(m) - F$ contains $M_2(m)$ as a subgraph, and hence $\mathcal{G}_{k,d}(m)$ is a k -FT graph for $M_2(m)$. \square

Next, we describe the construction of a k -FT graph $\mathcal{G}_{k,d}(m)$ for $M_2(m)$ in the case when $2^{h+2} - 2$ is not a divisor of m .

(C'1) Set $r = \lfloor m/(2^{h+2} - 2) \rfloor$ and $c = m$. Perform (C1), (C2), and (C3).

(C'2) For each $v \geq (r-1)c$, add $\gamma = N - rc(2^{h+2} - 2) + 1$ vertices, $a_{v,1}, a_{v,2}, \dots, a_{v,\gamma}$, and $\gamma + 1$ edges, $(a_{v,0}, a_{v,1}), \dots, (a_{v,\gamma}, a_{v,\gamma+1})$, to \mathcal{Q}_v in $\mathcal{G}_{k,d}(m')$, where $a_{v,0}$ and $a_{v,\gamma+1}$ are a root of Q_v^0 and that of Q_v^1 , respectively (See Fig.4);

(C'3) If $(u, v) \in E_{\text{hor}}(G_{N,t,d}^{r,c})$ and $u, v \geq (r-1)c$ then add γ edges, $(a_{u,x}, a_{v,x})$ ($x = 1, 2, \dots, \gamma$), to the graph obtained in (C'2).

Lemmas 4, 5, and 6 hold even in the case when $2^{h+2} - 2$ is not a divisor of m by a similar argument as the proofs in the case when $2^{h+2} - 2$ is a divisor of m . Hence, we have the following theorem.

Theorem 3. $\mathcal{G}_{k,d}(m)$ is a k -FT graph for $M_2(m)$ with $|V(\mathcal{G}_{k,d}(m))| = m^2 + O(d^2 k^{2+(2/d)})$ and $\Delta(\mathcal{G}_{k,d}(m)) = O(d^2)$.

Theorem 4. For any positive integers m, k and any constant $\varepsilon > 0$, we can construct a k -FT graph for $M_2(m)$ with $n + O(k^{2+\varepsilon})$ vertices and a constant maximum vertex degree, where $n = |V(M_2(m))| = m^2$.

Proof. The theorem follows from Theorem 3 by putting $d = \lceil 2/\varepsilon \rceil$. \square

4 Concluding Remarks

Since a k -FT graph $\mathcal{P}_{n,k}$ for an n -vertex path P_n with $n + O(k)$ vertices and maximum degree of 3 is shown in [14], it is possible to construct a k -FT graph for $M_2(m)$ with $m^2 + O(k^2)$ vertices and a constant maximum degree if $k = \Omega(m)$. In fact, the following graph $\mathcal{G}'_k(m)$ is a desired graph:

$$\begin{aligned} V(\mathcal{G}'_k(m)) &= V(\mathcal{P}_{m,k}) \times P_m \\ E(\mathcal{G}'_k(m)) &= \{([u_1, v], [u_2, v]) : (u_1, u_2) \in E(\mathcal{P}_{m,k})\} \\ &\quad \cup \{([u, v_1], [u, v_2]) : (v_1, v_2) \in E(P_m)\}. \end{aligned}$$

However, it still remains open whether it is possible to construct a k -FT graph for $M_2(m)$ with $m^2 + O(k^2)$ vertices and constant maximum degree for any $k = o(m)$.

References

1. N. Alon and F. Chung. Explicit construction of linear sized tolerant networks. *Discrete Math.*, 72:15–19, 1988.
2. J. Bruck, R. Cypher, and C. Ho. Fault-tolerant meshes and hypercubes with minimal numbers of spares. *IEEE Trans. on Comput.*, pp. 1089–1104, 1993.
3. J. Bruck, R. Cypher, and C. Ho. Tolerating faults in a mesh with a row of spare nodes. *Theoretical Computer Science*, 128(1-2):241–252, June 1994.
4. J. Bruck, R. Cypher, and C. Ho. Fault-tolerant meshes with small degree. *SIAM Journal on Computing*, 26:1764–1784, 1997.
5. S. Dutt and J. Hayes. Designing fault-tolerant systems using automorphisms. *Journal of Parallel and Distributed Computing*, 12:249–268, 1991.
6. P. Erdős, R. Graham, and E. Szemerédi. On sparse graphs with dense long paths. *Comp. and Math. with Appl.*, 1:145–161, 1975.
7. A. Farrag. New algorithm for constructing fault-tolerant solutions of the circulant graph configuration. *Parallel Computing*, 22:1239–1254, 1996.
8. F. Harary and J. Hayes. Node fault tolerance in graphs. *Networks*, 27:19–23, 1996.
9. J. Hayes. A graph model for fault-tolerant computing systems. *IEEE Trans. on Comput.*, C-25:875–883, 1976.
10. M. Paoli, W. Wong, and C. Wong. Minimum k-hamiltonian graphs, II. *J. Graph Theory*, 10:79–95, 1986.
11. H. Tamaki. Construction of the mesh and the torus tolerating a large number of faults. In *Proc. 6th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '94)*, pp. 268–277, 1994.
12. S. Ueno, A. Bagchi, S. Hakimi, and E. Schmeichel. On minimum fault-tolerant networks. *SIAM J. on Discrete Mathematics*, 6(4):565–574, November 1993.
13. W. Wong and C. Wong. Minimum k-hamiltonian graphs. *J. Graph Theory*, 8:155–165, 1984.
14. T. Yamada and S. Ueno. Optimal fault-tolerant linear arrays. In *Proc. 15th ACM SPAA*, pp. 60–64, 2003.
15. L. Zhang. Fault tolerant networks with small degree. In *Proceedings of Twelfth annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 64–69, 2000.
16. L. Zhang. Fault-tolerant meshes with small degree. *IEEE Transactions on Computers*, 51(5):553–560, May 2002.

Fault Hamiltonicity of Meshes with Two Wraparound Edges^{*}

Kyoung-Wook Park¹, Hyeong-Seok Lim¹,
Jung-Heum Park², and Hee-Chul Kim³

¹ Department of Computer Science, Chonnam National University,
300 Yongbong-dong, Buk-gu, Gwangju 500-757, Korea
kwpark@csblue.chonnam.ac.kr, hslim@chonnam.ac.kr

² School of Computer Science and Information Engineering,
The Catholic University of Korea
j.h.park@catholic.ac.kr

³ School of Computer Information and Communications Engineering,
Hankuk University of Foreign Studies
hckim@hufs.ac.kr

Abstract. We consider the fault hamiltonian properties of $m \times n$ meshes with two wraparound edges in the first row and the last row, denoted by $M_2(m, n)$, $m \geq 2$, $n \geq 3$. $M_2(m, n)$ is a spanning subgraph of $P_m \times C_n$ which has interesting fault hamiltonian properties. We show that $M_2(m, n)$ with odd n is hamiltonian-connected and 1-fault hamiltonian. For even n , $M_2(m, n)$, which is bipartite, with a single faulty element is shown to be 1-fault strongly hamiltonian-laceable. In previous works[1, 2], it was shown that $P_m \times C_n$ also has these hamiltonian properties. Our result shows that two additional wraparound edges are sufficient for an $m \times n$ mesh to have such properties rather than m wraparound edges. As an application of fault-hamiltonicity of $M_2(m, n)$, we show that the n -dimensional hypercube is strongly hamiltonian laceable if there are at most $n - 2$ faulty elements and at most one faulty vertex.

1 Introduction

Meshes represent the communication structures of many applications in scientific computations as well as the topologies of many large-scale interconnection networks. One of the central issues in parallel processing is embedding of linear arrays and rings into a faulty interconnection network. The embedding is closely related to a hamiltonian problem in graph theory.

An interconnection network is often modeled as an undirected graph, in which vertices and edges correspond to nodes and links, respectively. A graph G is called *hamiltonian-connected* if there exists a hamiltonian path joining every pair of vertices in G . We consider the hamiltonian properties of a graph in the presence of faulty elements(vertices and/or edges). A graph G is called *k-fault hamiltonian*

^{*} This work was supported by grant No. R01-2003-000-11676-0 from the Basic Research Program of the Korea Science & Engineering Foundation.

(resp. *k*-fault hamiltonian-connected) if $G - F$ has a hamiltonian cycle (resp. a hamiltonian path joining every pair of vertices) for any set F of faulty elements such that $|F| \leq k$. Apparently, a bipartite graph is not hamiltonian-connected. In [3], the concept of hamiltonian laceability for hamiltonian bipartite graphs was introduced. Bipartition sets of a bipartite graph are often represented as sets of black and white vertices. A bipartite graph G is *hamiltonian-laceable* if there is a hamiltonian path joining every pair of vertices with different colors. In [4], this concept was extended into strongly hamiltonian laceability. A hamiltonian laceable graph G with N vertices is *strong* if there is a path of length $N - 2$ joining every pair of vertices with the same color.

For any faulty set F such that $|F| \leq k$, a bipartite graph G which has an L^{opt} -path joining every pair of fault-free vertices is called *k-fault strongly hamiltonian laceable*[2]. An L^{opt} -path is defined as follows. Let G be a bipartite graph and let B and W be the sets of black and white vertices in G , respectively. Denote by F_v and F_e the sets of faulty vertices and edges in G , respectively. Let $F = F_v \cup F_e$, $f_v = |F_v|$, $f_e = |F_e|$, and $f = |F|$. The numbers of fault-free black and white vertices are denoted by n_b and n_w , respectively. When $n_b = n_w$, a fault-free path of length $2n_b - 1$ joining a pair of vertices with different colors is called an L^{opt} -path. For a pair of vertices with the same color, a fault-free path of length $2n_b - 2$ between them is called an L^{opt} -path. When $n_b > n_w$, fault-free paths of length $2n_w$ for a pair of black vertices, of length $2n_w - 1$ for a pair of vertices with different colors, and of length $2n_w - 2$ for a pair of white vertices, are called L^{opt} -paths. Similary, an L^{opt} -path can be defined when $n_w > n_b$. A fault-free cycle of length $2 \cdot \min\{n_b, n_w\}$ is called an L^{opt} -cycle. The lengths of an L^{opt} -path and an L^{opt} -cycle are the largest possible.

Fault hamiltonicity of various interconnection networks has been investigated. In [5] and [6], linear-time algorithms that find hamiltonian paths in $m \times n$ meshes were developed. In [7] and [8], the fault hamiltonian properties of $m \times n$ torus and $P_m \times C_n$ were considered, where $P_m \times C_n$ is a graph obtained by product of a path P_m with m vertices and a cycle C_n with n vertices. $P_m \times C_n$ forms an $m \times n$ mesh with a wraparound edge in each row. Futhermore, it was shown that $P_m \times C_n$ is hamiltonian-connected and 1-fault hamiltonian if it is not bipartite[1]; otherwise, $P_m \times C_n$ is 1-fault strongly hamiltonian laceable[2].

In this paper, we consider the hamiltonian properties of $m \times n$ mesh ($m \geq 2, n \geq 3$) with two wraparound edges in the first row and the last row. We denote the graph by $M_2(m, n)$. We show that $M_2(m, n)$ with odd n is hamiltonian-connected and 1-fault hamiltonian. For a graph G to be hamiltonian-connected, G should be non-bipartite and $\delta(G) \geq 3$, where $\delta(G)$ is the minimum degree of G . For a graph G to be *k*-fault hamiltonian, it is necessary that $k \leq \delta(G) - 2$. Thus, $M_2(m, n)$ with odd n satisfies the above condition by adding two(minimum) edges to an $m \times n$ mesh. Futhermore, for n even, we show that $M_2(m, n)$, which is bipartite, with a single faulty element is strongly hamiltonian laceable. In previous works[1, 2], it was shown that $P_m \times C_n$ also has these hamiltonian properties. Our result shows that two additional wraparound edges are sufficient for an $m \times n$ mesh to have such properties rather than m wraparound edges.

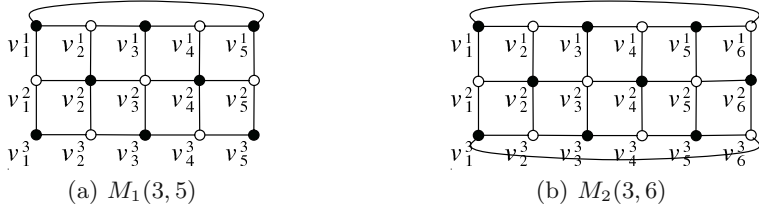


Fig. 1. Examples of $M_1(m, n)$ and $M_2(m, n)$

For some m and n , $M_2(m, n)$ is a spanning subgraph of many interconnection networks such as tori, hypercubes, k -ary n -cubes, double loop networks, and recursive circulants. Thus, our results can be applied to discover the fault hamiltonicity of such interconnection networks. It was shown in [9] that n -dimensional hypercube Q_n with $n - 2$ faulty edges is strongly hamiltonian laceable. By applying fault hamiltonicity of $M_2(m, n)$, we show that Q_n with at most $n - 2$ faulty elements and at most one faulty vertex is strongly hamiltonian laceable.

2 Preliminaries

Let $M(m, n) = (V, E)$ be an $m \times n$ mesh, where the vertex set V is $\{v_j^i | 1 \leq i \leq m, 1 \leq j \leq n\}$ and the edge set E is $\{(v_j^i, v_{j+1}^i) | 1 \leq i \leq m, 1 \leq j < n\} \cup \{(v_j^i, v_j^{i+1}) | 1 \leq i < m, 1 \leq j \leq n\}$. We propose a graph which has two wraparound edges in the first row and the last row in $M(m, n)$.

Definition 1. Let $M(m, n) = (V, E)$. $M_2(m, n)$ is defined as (V_{M_2}, E_{M_2}) , where the vertex set $V_{M_2} = V$ and the edge set $E_{M_2} = E \cup \{(v_1^1, v_n^1), (v_1^m, v_n^m)\}$.

The vertices of $M_2(m, n)$ are colored with black and white as follows: v_j^i is called a *black* vertex if $i + j$ is even; otherwise it is a *white* vertex. We denote by $R(i)$ and $C(j)$ the vertices in row i and column j , respectively. That is, $R(i) = \{v_j^i | 1 \leq j \leq n\}$ and $C(j) = \{v_j^i | 1 \leq i \leq m\}$. We let $R(i : j) = \cup_{i \leq k \leq j} R(k)$ if $i \leq j$; otherwise $R(i : j) = \emptyset$. Similarly, $C(i : j) = \cup_{i \leq k \leq j} C(k)$ if $i \leq j$; otherwise $C(i : j) = \emptyset$.

We denote by $H[s, t|X]$ a hamiltonian path from s to t in the subgraph $G \langle X \rangle$ induced by a vertex set X , if any. A path is represented as a sequence of vertices. If X is an empty set, $H[s, t|X]$ is an empty sequence. We denote by $v_j^i \rightarrow v_{j'}^i$ a path $(v_j^i, v_{j+1}^i, \dots, v_{j'-1}^i, v_{j'}^i)$ if $j < j'$; otherwise, $(v_j^i, v_{j-1}^i, \dots, v_{j'+1}^i, v_j^i)$. Similarly, $v_j^i \rightarrow v_{j'}^{i'}$ a path from v_j^i to $v_{j'}^{i'}$ in the subgraph $G \langle C(j) \rangle$. We employ three lemmas on the hamiltonian properties of $M(m, n)$ and $P_m \times C_n$. We call a vertex in a mesh a *corner vertex* if it is of degree two.

Lemma 1. [10] (a) If mn is even, then $M(m, n)$ has a hamiltonian path from any corner vertex v to any other vertex with color different from v . (b) If mn is odd, then $M(m, n)$ has a hamiltonian path from any corner vertex v to any other vertex with the same color as v .

Lemma 2. [5] *Let two vertices s, t have different color each other. (a) If $m, n \geq 4$ and mn is even, then $M(m, n)$ has a hamiltonian path joining s and t . (b) If $m = 2$, $n \geq 3$, and $s, t \notin C(k)(2 \leq k \leq n - 1)$, then $M(m, n)$ has a hamiltonian path joining s and t .*

Lemma 3. (a) *For $m \geq 2, n \geq 3$ odd, $P_m \times C_n$ is hamiltonian-connected and 1-fault hamiltonian[1].* (b) *For $m \geq 2, n \geq 4$ even, $P_m \times C_n$ is 1-fault strongly hamiltonian-laceable[2].*

Let P and Q be two vertex-disjoint paths (a_1, a_2, \dots, a_k) and (b_1, b_2, \dots, b_l) in a graph G , respectively, such that (a_i, b_1) and (a_{i+1}, b_l) are edges in G . If we replace (a_i, a_{i+1}) with (a_i, b_1) and (a_{i+1}, b_l) , then P and Q are merged into a single path $(a_1, a_2, \dots, a_i, b_1, b_2, \dots, b_l, a_{i+1}, \dots, a_k)$. We call such a replacement a *merge* of P and Q w.r.t. (a_i, b_1) and (a_{i+1}, b_l) . If P is a closed path (that is, a cycle), the merge operation results in a single cycle. We denote by $V(P)$ the set of vertices on a path P .

To show the fault hamiltonicity of $M_2(m, n)$, we first show some hamiltonian properties of $M_1(m, n)$ which has a single wraparound edge on the first row in $M(m, n)$. An $M_1(m, n)$ has two corner vertices v_1^m and v_n^m .

Lemma 4. *For $m \geq 2, n \geq 3$ odd, $M_1(m, n)$ has a hamiltonian path between any corner vertex s and any other vertex t .*

Proof. The proof is by induction on m . Without loss of generality, we assume that $s = v_1^m$. First, we observe that the lemma holds for $m = 2$. For $t \in B$, there exists a hamiltonian path by Lemma 1; otherwise we can construct a hamiltonian path $P = (s, v_1^1, H[v_n^1, t|C(2 : n)])$. By Lemma 1, $H[v_n^1, t|C(2 : n)]$ exists.

For $m \geq 3$, we assume that the lemma is true for every $k < m$. The proof is divided into two cases.

Case 1: $t \in R(1 : m - 1)$. When $t \neq v_n^{m-1}$, we can construct a hamiltonian path $P = (s, v_2^m \rightarrow v_n^m, H[v_n^{m-1}, t|R(1 : m - 1)])$. By induction hypothesis, $H[v_n^{m-1}, t|R(1 : m - 1)]$ exists. When $t = v_n^{m-1}$, $P = (s, v_1^{m-1} \rightarrow v_1^1, H[s', t|C(2 : n)])$, where s' is v_n^1 if m is odd; s' is v_2^1 if m is even.

Case 2: $t \in R(m)$. Let $t = v_i^m$. By induction hypothesis, there exists a hamiltonian path P' joining v_{i-1}^{m-1} and v_n^{m-1} in $G \setminus R(1 : m - 1)$. We construct a hamiltonian path $P = (s, v_2^m \rightarrow v_{i-1}^m, P', v_n^m, v_{n-1}^m \rightarrow t)$. \square

In the following lemmas, we just summarize the results and omit the proofs.

Lemma 5. *For $m \geq 2, n \geq 4$ even, $M_1(m, n)$ is strongly hamiltonian laceable.*

Lemma 6. *For $m \geq 2, n \geq 4$ even, $M_1(m, n)$ with a single faulty vertex v_f has an L^{opt} -path joining a corner vertex s and any other vertex t if s has a different color from v_f and at most one of v_f and t is adjacent to s .*

3 Hamiltonian Properties of $M_2(m, n)$

3.1 $M_2(m, n)$ with Odd n

When n is odd, $M_2(m, n)$ is not bipartite. We show that $M_2(m, n)$ is hamiltonian-connected and 1-fault hamiltonian.

Theorem 1. For $m \geq 2, n \geq 3$ odd, $M_2(m, n)$ is hamiltonian-connected.

Proof. The proof is by induction on m . $M_2(2, n)$ is isomorphic to $P_2 \times C_n$. Thus, the theorem is true by Lemma 3 when $m = 2$. For $m \geq 3$, we assume that the theorem is true for every $k < m$. Let $s = v_i^x, t = v_j^y$. We show that $M_2(m, n)$ has a hamiltonian path P between s and t . The proof is divided into two cases.

Case 1: $s, t \in R(1 : m - 1)$. If we assume that a virtual edge (v_1^{m-1}, v_n^{m-1}) exists, then there exists a hamiltonian path P' joining s and t in $G \langle R(1 : m - 1) \rangle$ by induction hypothesis. If P' passes through the edge (v_1^{m-1}, v_n^{m-1}) , then we can construct a hamiltonian path P by replacing (v_1^{m-1}, v_n^{m-1}) with a path $(v_1^{m-1}, v_1^m \rightarrow v_n^m, v_n^{m-1})$; otherwise we choose an edge (u, v) in $G \langle R(m - 1) \rangle$ such that P' includes it. Let u' and v' be the vertices in $R(m)$ adjacent to u and v , respectively. Since $\langle R(m) \rangle$ forms a cycle, it has a hamiltonian path P'' joining u' and v' . By a merge of P' and P'' w.r.t. (u, u') and (v, v') , we have a hamiltonian path.

Case 2: $s \in R(1 : m - 1)$ and $t \in R(m)$. When $s \in R(2 : m - 1)$, this case is symmetric to Case 1. Thus, we only consider the case that $s \in R(1)$.

Case 2.1: $m = 3$. If either s or t is on the first column or the last column, then there exist a hamiltonian path by Lemma 4. Otherwise (that is, $s, t \in C(2 : n - 1)$), by Lemma 1 and Lemma 2, we can construct a hamiltonian path P as follows:

- (i) $s, t \in W, P = (s, v_{i+1}^1 \rightarrow v_n^1, v_1^1 \rightarrow v_{i-1}^1, H[v_{i-1}^2, v_1^3 | R(2 : 3)] \cap C(1 : j - 1)), H[v_n^3, t | R(2 : 3) \cap C(j : n)])$.
- (ii) $s \in B$ and $t \in W, P = (H[s, v_1^3 | C(1 : i)], H[v_n^3, t | C(i + 1 : n)])$.
- (iii) $s, t \in B$,

$$P = \begin{cases} (H[s, v_{i+1}^1 | C(1 : i + 1)], H[v_{i+2}^1, t | C(i + 2 : n)]) & \text{if } i \neq j \\ (s, H[v_{i-1}^1, v_{i-1}^2 | C(1 : i - 1)], v_i^2, H[v_{i+1}^2, v_{i+1}^3 | C(i + 1 : n)], t) & \text{if } i = j \end{cases}$$

Case 2.2: $m \geq 4$. By Lemma 4, there exist two paths $P' = (H[s, v_1^{m-2} | R(1 : m - 2)])$ and $P'' = (H[v_1^{n-1}, t | R(m - 1 : m)])$. (P', P'') forms a hamiltonian path. \square

Theorem 2. For $m \geq 2, n \geq 3$ odd, $M_2(m, n)$ is 1-fault hamiltonian.

Proof. We prove by induction on m . Due to Lemma 3, the theorem holds for $m = 2$. For $m \geq 3$, we assume that the theorem is true for every $k < m$, and we consider $M_2(m, n)$. Without loss of generality, we assume that the faulty element is contained in $G \langle R(1 : m - 1) \rangle$.

If we assume that a virtual edge (v_1^{m-1}, v_n^{m-1}) exists, then there exists a fault-free hamiltonian cycle C' in $G \langle R(1 : m - 1) \rangle$ by induction hypothesis. If C' passes through (v_1^{m-1}, v_n^{m-1}) , then we can construct a hamiltonian cycle C by replacing (v_1^{m-1}, v_n^{m-1}) with a path $(v_1^{m-1}, v_1^m \rightarrow v_n^m, v_n^{m-1})$; If C' does not pass through (v_1^{m-1}, v_n^{m-1}) , we choose an edge (u, v) in $G \langle R(m - 1) \rangle$ such that C' includes it. Let u' and v' be the vertices in $R(m)$ adjacent to u and v , respectively. Since $G \langle R(m) \rangle$ forms a cycle, it has a hamiltonian path P' joining u' and v' . By a merge of $C' - (u, v)$ and P' w.r.t. (u, u') and (v, v') , we have a fault-free hamiltonian cycle C . \square

3.2 $M_2(m, n)$ with Even n

When n is even, $M_2(m, n)$ is bipartite. First, we show that $M_2(m, n)$ with a single faulty vertex is strongly hamiltonian-laceable.

Lemma 7. *For $n \geq 4$ even, $M_2(3, n)$ with a single faulty vertex is strongly hamiltonian laceable.*

Proof. L^{opt} -paths can be constructed for all cases: i) $s, t \in R(1 : 2)$, ii) $s \in R(1 : 2), t \in R(3)$, iii) $s, t \in R(3)$. The details are omitted. \square

Lemma 8. *For $m \geq 2, n \geq 4$ even, $M_2(m, n)$ with a single faulty vertex is strongly hamiltonian laceable.*

Proof. The proof is by induction on m . For $m = 2$ and $m = 3$, the lemma is true by Lemma 3 and Lemma 7, respectively. For $m \geq 4$, we assume that the lemma holds for every $k < m$, and we consider $M_2(m, n)$. Let $s = v_i^x, t = v_j^y$. Without loss of generality, we assume that a faulty vertex $v_f \in W$ and $v_f \in R(1 : m - 2)$.

Case 1: $s, t \in R(1 : m - 1)$. Similar to Case 1 in Theorem 1, we can construct an L^{opt} -path.

Case 2: $s \in R(1 : m - 1)$ and $t \in R(m)$.

Case 2.1: $s \in R(1 : m - 2)$. We choose a black vertex s' which is one of the two vertices v_1^{m-2} and v_n^{m-2} . If $s = s'$ or s' is adjacent to both v_f and s , then let s' be the black vertex in $R(m - 2) \cap C(2 : n - 1)$. There exists an L^{opt} -path P' joining s and s' in $G \langle R(1 : m - 2) \rangle$ as follows. When either s or s' is v_n^{m-2} (resp. v_1^{m-2}) and m is even (resp. odd), P' exists by Lemma 6. Otherwise P' can be constructed as follows:

$$P' = \begin{cases} (H[s, v_{n-1}^1 | R(1 : m - 3) \cap C(n - 1 : n)], \\ H[v_{n-2}^1, s' | R(1 : m - 2) \cap C(1 : n - 2)]) & \text{if } m \text{ is even} \\ (H[s, v_1^1 | R(1 : m - 3) \cap C(1 : 2)], \\ H[v_n^1, s' | R(1 : m - 2) \cap C(3 : n)]) & \text{if } m \text{ is odd} \end{cases}$$

Let t' be the vertex in $R(m - 1)$ adjacent to s' . By Lemma 5, there exists an L^{opt} -path P'' joining t' and t in $G \langle R(m - 1 : m) \rangle$. P' and P'' form an L^{opt} -path.

Case 2.2: $s \in R(m - 1)$. If v_f is in $R(2 : m - 2)$, then this case is symmetric to Case 1. Thus, we only consider the case that $v_f \in R(1)$. We choose two vertices u and v as follows. When $s, t \in B$, let $u = v_2^{m-2}$ and $v = v_n^{m-2}$ if m is even; otherwise $u = v_1^{m-2}$ and $v = v_{n-1}^{m-2}$. When at least one of s or t is white, $u = v_1^{m-2}$ and $v = v_n^{m-2}$. Let u' and v' be the vertices in $R(m - 1)$ adjacent to u and v , respectively. By Lemma 6, there exists an L^{opt} -path P' joining u and v in $G \langle R(1 : m - 2) \rangle$. Let P'' and P''' be two vertex-disjoint paths in $G \langle R(m - 1 : m) \rangle$ such that $V(P'') \cup V(P''') = R(m - 1 : m)$ and they are joining s and u', v' and t , or s and v', u' and t , respectively. We can construct an L^{opt} -path $P = (P'', P', P''')$. P'' and P''' can be constructed as follows:

Without loss of generality, we assume that m is even.

Case 2.2.1 $i \leq j$. (i) $s, t \in B$,

$$\begin{aligned} P'' &= \begin{cases} (s, u') & \text{if } t \in C(2) \\ (H[s, u'|C(1:i) \cap R(m-1:m)]) & \text{if } t \in C(3:n) \end{cases} \\ P''' &= \begin{cases} (H[v', v_n^m|C(3:n) \cap R(m-1:m)], v_1^m, t) & \text{if } t \in C(2) \\ (H[v', t|C(i+1:n) \cap R(m-1:n)]) & \text{if } t \in C(3:n) \end{cases} \end{aligned}$$

(ii) $s \in W$ and $t \in B$, $P'' = (H[s, u'|C(1:i) \cap R(m-1:m)])$ and

$$P''' = (H[v', t|C(i+1:n) \cap R(m-1:m)]).$$

(iii) $s \in B$ and $t \in W$, $P'' = (s, v_{i-1}^{m-1} \rightarrow u')$ and

$$P''' = (H[v', v_n^m|C(j+1:n) \cap R(m-1:m)], v_1^m \rightarrow v_i^m, H[v_{i+1}^m, t|C(i+1:j) \cap R(m-1:m)]).$$

(iv) $s, t \in W$. $P'' = (H[s, v'|C(1:j-1) \cap R(m-1:m)])$ and

$$P''' = (H[v', v_{j+1}^y|C(j+1:n) \cap R(m-1:m)], t).$$

Case 2.2.2 $i > j$. Similar to Case 2.2.1, P'' and P''' can be constructed. The details of P'' and P''' are omitted.

Case 3: $s, t \in R(m)$. If v_f is in $R(2:m-2)$, then this case is symmetric to Case 1. Thus, we only consider the case that $v_f \in R(1)$. The same way as Case 2.2, we can construct an L^{opt} -path P except the case that $s \in B$ and $t \in W$. We only show the case that $s \in B$ and $t \in W$. An L^{opt} -path P' in $G \langle R(1:m-2) \rangle$ can be obtained by using the same way as Case 2.2, and two vertex-disjoint paths P'' and P''' in $G \langle R(m-1:m) \rangle$ can be constructed as follows: $P'' = (H[s, v_{j-1}^{m-1}|C(i:j-1) \cap R(m-1:m)], v_j^{m-1} \rightarrow v')$ and $P''' = (H[u', v_1^m|C(1:i-1) \cap R(m-1:m)], v_n^m \rightarrow t)$. (P'', P', P''') forms an L^{opt} -path. \square

Lemma 9. For $m \geq 2, n \geq 4$ even, $M_2(m, n)$ with a single faulty edge is hamiltonian laceable.

Proof. We prove by induction on m . Due to Lemma 3, the lemma holds for $m = 2$. For $m \geq 3$, we assume that the lemma is true for every $k < m$, and we consider $M_2(m, n)$. There exists a hamiltonian path by Lemma 2, if e_f is one of (v_1^1, v_n^1) and (v_1^m, v_n^m) . Let $s = v_i^x$ and $t = v_j^y$. Without loss of generality, we assume that the faulty edge $e_f \in G \langle R(1:m-1) \rangle$.

Case 1: $s, t \in R(1:m-1)$. Similary to Case 1 in Theorem 1, we can construct an L^{opt} path.

Case 2: $s \in R(1:m-1)$ and $t \in R(m)$.

Case 2.1: $e_f = (v_k^z, v_k^{z+1})$. When $x > z$, there exists a hamiltonian path P' joining s and t in $G \langle R(z+1:m) \rangle$ by Lemma 5. We choose an edge (u, v) in $G \langle R(z+1) \rangle$ such that P' includes it and neither u nor v is v_k^{z+1} . Let u' and v' be the vertices in $R(z)$ adjacent to u and v , respectively. By Lemma 5, there exists a hamiltonian path P'' joining u and v in $G \langle R(1:z) \rangle$. By a merge of P' and P'' w.r.t. (u, u') and (v, v') , we have a hamiltonian path P . When $x \leq z$, we choose a vertex s' in $R(z)$ such that s' has a different color from s and is

not v_k^z . Let t' be the vertex in $G \langle R(z+1) \rangle$ adjacent to s' . By Lemma 5, we can construct a hamiltonian path $P = (H[s, s'|R(1 : z)], H[t', t|R(z+1 : m)])$.

Case 2.2: $e_f = (v_k^z, v_{k+1}^z)$. Without loss of generality, P can be constructed according to the three cases. (i) $i \leq k < j$, (ii) $i \leq j \leq k$, and (iii) $k < i \leq j$. We only show the case that $i \leq k < j$, $s \in B$, and $t \in W$. Proofs of other cases are omitted.

a) For $k = 1$, If $x > z$, then $s \in B$ and $s \in C(1)$.

$$P = (s \rightarrow v_1^1, H[v_n^1, v_{j-1}^x | C(2 : n) \cap R(1 : x)], H[v_{j-1}^{x+1}, t | R(x+1 : m)])$$

If $x \leq z$,

$$P = \begin{cases} (s \rightarrow v_1^m, v_2^m \rightarrow v_2^x, H[v_2^{x-1}, v_3^{x-1} | R(1 : x-1)], \\ H[v_3^x, t | C(3 : n) \cap R(x : m)]) & \text{if } j = n \\ (s \rightarrow v_1^m, v_n^m \rightarrow v_n^x, H[v_n^{x-1}, v_{n-1}^{x-1} | R(1 : x-1)], \\ H[v_{n-1}^x, t | C(2 : n-1) \cap R(x : m)]) & \text{if } j \neq n \end{cases}$$

b) For $1 < k < n-1$,

$$P = \begin{cases} (H[s, v_k^m | C(1 : k)], H[v_{k+1}^m, t | C(k+1 : n)]) & \text{if } j = n \\ (H[s, v_1^m | C(1 : k)], H[v_n^m, t | C(k+1 : n)]) & \text{if } j \neq n \end{cases}$$

Case 3: $s, t \in R(m)$.

Case 3.1: $e_f = (v_k^z, v_{k+1}^{z+1})$. By Lemma 5, there exists a hamiltonian path P' joining s and t in $G \langle R(z+1 : m) \rangle$. We choose an edge (u, v) in $G \langle R(z+1) \rangle$ such that P' includes it and neither u nor v is v_k^{z+1} . Let u' and v' be the vertices in $R(z)$ adjacent to u and v , respectively. In $G \langle R(1 : z) \rangle$, there exists a hamiltonian path P'' joining u' and v' by Lemma 5. By a merge of P' and P'' w.r.t. (u, u') and (v, v') , we have a hamiltonian path.

Case 3.2: $e_f = (v_k^z, v_{k+1}^z)$. Without loss of generality, we assume that $i < j$.

Case 3.2.1: $i < j \leq k$. When $s \in B$ and $t \in W$,

$$P = \begin{cases} (H[s, v_1^m | C(1 : j-1) \cap R(2 : m)], H[v_n^m, v_n^1 | C(k+1 : n)], \\ v_1^1 \rightarrow v_{j-1}^1, H[v_j^1, t | C(j : k)]) & \text{if } m \text{ is even} \\ (H[s, v_{j-1}^2 | C(1 : j-1) \cap R(2 : m)], \\ H[v_j^2, v_j^1 | C(j : k) \cap R(1 : m-1)], v_{j-1}^1 \rightarrow v_1^1, \\ H[v_n^1, v_{k+1}^m | C(k+1 : n)], v_k^m \rightarrow t) & \text{if } m \text{ is odd} \end{cases}$$

When $s \in W$ and $t \in B$,

$$P = \begin{cases} (s \rightarrow v_1^m, H[v_n^m, v_n^1 | C(k+1 : n)], \\ H[v_1^1, v_1^{m-2} | C(1 : k) \cap R(1 : m-2)], v_1^{m-1} \rightarrow v_i^{m-1}, \\ H[v_{i+1}^{m-1}, t | C(i+1 : k) \cap R(m-1 : m)]) & \text{if } m \text{ is even} \\ (H[s, v_{j-1}^{m-1} | C(1 : j-1) \cap R(m-1 : m)], v_j^{m-1} \rightarrow v_k^{m-1}, \\ H[v_k^{m-2}, v_1^1 | C(1 : k) \cap R(1 : m-2)], \\ H[v_n^1, v_{k+1}^m | C(k+1 : n)], v_k^m \rightarrow t) & \text{if } m \text{ is odd} \end{cases}$$

Case 3.2.2: $i \leq k < j$. We can construct a hamiltonian path P as follows. When m is even, let $s' = v_1^m$ and $t' = v_n^m$ if $s \in B$; $s' = v_1^1$ and $t' = v_n^1$ if $s \in W$.

When m is odd and $s \in B$, let $s' = v_k^m$ and $t' = v_{k+1}^m$ if k is even; $s' = v_1^1$ and $t' = v_n^1$ if k is odd. $P = (H[s, s'|C(1 : k)], H[t', t|C(k + 1 : n)])$. When m is odd and $s \in W$, $P = (H[s, v_k^m|C(i : k)], v_{k+1}^m \rightarrow v_{k+1}^1 \rightarrow v_n^1, H[v_1^1, v_n^1|C(1 : i - 1)], H[v_n^m, t|C(k + 2 : n) \cap R(2 : m)])$. \square

Theorem 3. For $m \geq 2, n \geq 4$ even, $M_2(m, n)$ is 1-fault strongly hamiltonian laceable.

Proof. By Lemma 8, $M_2(m, n)$ with a single faulty vertex is strongly hamiltonian laceable. By Lemma 9, $M_2(m, n)$ with a single faulty edge has a hamiltonian path between any two vertices with different colors. It remains to show that there is an L^{opt} -path (of length $mn - 2$) joining every pair of vertices s and t with the same color. Let (u, v) be the faulty edge. Without loss of generality, we assume that $u \in B$ and $v \in W$. When s and t are black, we can find an L^{opt} -path P between s and t regarding v as a faulty vertex by using Lemma 8. P does not pass through (u, v) as well as v , and the length of P is $mn - 2$. Thus, P is a desired L^{opt} -path. In a similar way, we can construct an L^{opt} -path for a pair of white vertices. \square

4 Fault Hamiltonicity of Hypercubes

An n -dimensional hypercube, denoted by Q_n , consists of 2^n vertices that can be represented in the form of binary strings, $b_n b_{n-1} \dots b_1$. Two vertices are adjacent if and only if their labels differ in exactly one bit. An edge is referred to as an i -dimension edge if the vertices it connects differ in bit position i . A k -dimensional subcube in Q_n is represented by a string of n symbols over set $\{0, 1, *\}$, where $*$ is a *don't care* symbol, such that there are exactly k $*$'s in the string.

Lemma 10. For $f_e \leq n - 3$ and $1 \leq r \leq n - f_e - 2$, Q_n with f_e faulty edges has $M_2(2^r, 2^{n-r})$ as a spanning subgraph.

Proof. Let $D = \{1, 2, \dots, n\}$ be the set of dimensions in Q_n , $D_f = \{f_1, f_2, \dots, f_i\}$ be the set of dimensions which contain faulty edges, and $D_s = D - D_f = \{s_1, s_2, \dots, s_j\}$. Since $f_e \leq n - 3$, we have $|D_s| \geq 3$.

If we replace $b_{s_1}, b_{s_2}, \dots, b_{s_r}$ bits of each vertex label in Q_n by $*$, then Q_n can be partitioned into 2^{n-r} r -subcubes and these subcubes form Q_{n-r} by replacing each subcube as a vertex. We denote such a graph by a condensation graph Q_{n-r}^C of Q_n . Let u, v be the vertices in Q_n , and $\mathcal{C}_u, \mathcal{C}_v$ be the components containing u and v , respectively. We assume that an edge $(\mathcal{C}_u, \mathcal{C}_v)$ of Q_{n-r}^C is faulty if (u, v) is faulty. For all $n \geq 2$, Q_n with $n - 2$ faulty edges has a hamiltonian cycle [11]. Since $f_e \leq n - r - 2$, there exists a hamiltonian cycle in Q_{n-r}^C . Let $C = (x_1, x_2, \dots, x_d, \dots, x_{2^{n-r}}, x_1)$ be a hamiltonian cycle in Q_{n-r}^C , where $1 \leq d \leq 2^{n-r}$. Each vertex of Q_n can be mapped to v_d^k of $M_2(m, n)$ as follows:

In $b_n b_{n-1} \dots b_1$ of each vertex label of Q_n ,

- (i) $b_{s_1} b_{s_2} \dots b_{s_r}$ is the k -th sequence of r -bit Gray code, and
- (ii) $(n - r)$ -bits (except $b_{s_1}, b_{s_2}, \dots, b_{s_r}$ bits) represent the label of the d -th vertex in C .

Thus, Q_n has fault-free $M_2(2^r, 2^{n-r})$ as a spanning subgraph. \square

By applying fault-hamiltonicity of $M_2(m, n)$ to a hypercube, we have the following theorem.

Theorem 4. Q_n with $f \leq n - 2$ and $f_v \leq 1$ is strongly hamiltonian laceable.

5 Conclusion

In this paper, we considered the fault hamiltonian properties of $m \times n$ meshes with two wraparound edges in the first row and the last row. We showed that $M_2(m, n)$ with odd n is hamiltonian-connected and 1-fault hamiltonian. $M_2(m, n)$ has these hamiltonian properties by adding minimum edges to an $m \times n$ mesh. For $n \geq 4$ even, we showed that $M_2(m, n)$ is 1-fault strongly hamiltonian laceable. By applying fault hamiltonicity of $M_2(m, n)$ to a hypercube, we obtained that Q_n with at most $n - 2$ faulty elements and at most one faulty vertex is strongly hamiltonian laceable. Also, our results can be applied to the fault hamiltonian properties of other interconnection networks which has $M_2(m, n)$ as a spanning subgraph.

References

1. C.-H. Tsai, J. M. Tan, Y. C. Chuang and L.-H. Hsu, Fault-free cycles and links in faulty recursive circulant graphs, Proceedings of the 2000 International Computer Symposium: Workshop on Computer Algorithms and Theory of Computation (2000) 74–77.
2. J.-H. Park and H.-C. Kim, Fault hamiltonicity of product graph of path and cycle, International Computing and Combinatorics Conference(COCOON) (2003) 319–328.
3. G. Simmons, Almost all n -dimensional rectangular lattices are Hamilton laceable, Congressus Numerantium **21** (1978) 103–108.
4. S. Y. Hsieh, G. H. Chen and C. W. Ho, Hamiltonian-laceability of star graphs, Networks **36** (2000) 225–232.
5. A. Itai, C. H. Papadimitriou and J. L. Czwarcfiter, Hamiltonian paths in grid graphs. SIAM Journal of Computing **11** (4) (1982) 676–686.
6. S. D. Chen, H. Shen and R. W. Topor, An efficient algorithm for constructing hamiltonian paths in meshes, Parallel Computing **28** (2002) 1293–1305.
7. J. S. Kim, S. R. Maeng and H. Yoon, Embedding of rings in 2-D meshes and tori with faulty nodes, Journal of Systems Architecture **43** (1997) 643–654.
8. M. Lewinter and W. Widulski, Hyper-hamilton laceable and caterpillar-spannable product graphs, Computer Math. Applic. **34** (11) (1997) 99–104.
9. C.-H. Tsai, J. M. Tan, T. Lian and L.-H. Hsu, Fault-tolerant hamiltonian laceability of hypercubes, Information Processing Letters **83** (2002) 301–306.
10. C. C. Chen and N. F. Quimpo, On strongly Hamiltonian abelian group graphs. Combinatorial Mathematics VIII. Lecture Notes in Mathematics **884** (1980) 23–34.
11. S. Latifi, S. Zheng, N. Bagherzadeh, Optimal ring embedding in hypercubes with faulty links, International Symposium on Fault-Tolerant Computing(FTCS) (1992) 178–184.

On the Expected Time for Herman's Probabilistic Self-stabilizing Algorithm^{*}

Toshio Nakata

Department of Information Education, Fukuoka University of Education,
Akama-Bunkyo-machi, Munakata, Fukuoka, 811-4192, Japan
nakata@fukuoka-edu.ac.jp

Abstract. In this article we investigate the expected time for Herman's probabilistic self-stabilizing algorithm in distributed systems: Suppose that the number of identical processes in a ring, say n , is odd and $n \geq 3$. If the initial configuration of the ring is not "legitimate", that is, the number of tokens differs from one, then execution of the algorithm made up of synchronous probabilistic procedures with a parameter $0 < r < 1$ results in convergence to a legitimate configuration with a unique token (Herman's algorithm). We then show that the expected time of the convergence is less than $\frac{\pi^2-8}{8r(1-r)}n^2$. Moreover there exists a configuration whose expected time is $\Theta(n^2)$. The method of the proof is based on the analysis of *coalescing random walks*.

Keywords: probabilistic self-stabilization, Herman's algorithm, random walk, coalescing time.

1 Introduction

Distributed systems based on self-stabilizing algorithms were introduced by Dijkstra [5]. These propose an elegant way of solving the problem of fault-tolerance (see also [6, 11]). In general, self-stabilizing algorithms have that arbitrary configuration reaches "legitimate" configuration satisfying desired property in finite times. Moreover probabilistic methods are often studied as solutions to problems for distributed computing in order to improve the efficiency of deterministic algorithms (see [6]). One of the useful probabilistic self-stabilizing algorithms was proposed by Herman [8]. The basic setting is the following: Suppose that the number of identical processes in a ring, say n , is odd and $n \geq 3$. If the initial configuration of the ring is not *legitimate*, that is, the number of tokens differs from one, then execution of the algorithm made up of synchronous probabilistic procedures with a parameter $0 < r < 1$ results in convergence to a legitimate configuration with a unique token (Herman's algorithm).

For the algorithm we then show that the expected time of the convergence is less than $\frac{\pi^2-8}{8r(1-r)}n^2$. Moreover we also give a configuration whose expected time

^{*} This work was supported by Grant-in-Aid for Young Scientists (B) No. 14740077 from MEXT Japan.

is $\Theta(n^2)$. The method of the proof is based on the analysis of the expected time of *coalescing random walks*.

A model of coalescing random walks was proposed by Aldous and Fill [2, Chapter 14]. They formalized them using *continuized* chains with 1-exponential distributions. Note that many quantities are unchanged by the passage from the discrete time chain to the continuized chain. Their setting is the following: Fix a connected graph G with order n . Consider independent random walks on each vertex. When some of them meet, they coalesce into a new random walk. What is the expected time that all walks coalesce into a single random walk? They showed that the time is less than $e(\log n + 2)T_G$, where T_G is the maximal (in the meaning of start and goal points) expected *hitting time* of G . Brightwell and Winkler [3] showed that T_G is bounded by $(4/27 + o(1))n^3$, which is attained by some lollipop graphs. Moreover Coppersmith et al. [4], Tetali and Winkler [13] studied the *meeting time*, which is the expected number of steps that two independent random walks collide on a connected graph. They proved that $M_G \leq (4/27 + o(1))n^3$, where M_G is the maximal (in the meaning of start points) meeting time on a connected graph G with order n . Furthermore, Hassin and Peleg [9] applied the results of meeting times to an analysis of the expected absorption time in distributed probabilistic polling. In their setting they pointed out that the time is equivalent to the expected time of coalescing random walks. They showed that the time is $O(M_G \log n)$. Their key idea of the estimate is the following: Since M_G is the maximal meeting time, at least $n/2$ random walks have to meet after the time M_G . By repetition the results can be obtained. Recently Adler et al. [1] showed good estimates of expected coalescing times for iid random variables without considering graph structures. They noted applications to population biology.

For expected times of Herman's self-stabilizing algorithm we also apply the method of coalescing random walks. We regard tokens in the ring as independent random walks. Though in our case random walks do not coalesce but disappear when two random walks collide, we can see that the expected times do not change essentially. Noting that both $M_{n\text{-ring}}$ and $T_{n\text{-ring}}$ are $\Theta(n^2)$, we can drop $\log n$ term for Aldous & Fill's/Hassin & Peleg's results. Moreover Aldous and Fill [2, Chapter 14 §3] conjectured that there exists an absolute constant K such that on any graph G the expected time of coalescing random walks is bounded by KT_G . Hence our estimate would be one of the positive example for their conjecture. To estimate the expected time of coalescing random walks, we are aiming at not the *maximal* meeting time but the *minimal* meeting time on the ring.

The plan of the paper as follows: After some preliminary about Herman's algorithm (Section 2), we state the estimate of the expected time of Herman's algorithm in Section 3.1. The proof is given in Section 3.2. Moreover showing future works, we conclude in Section 4. The rough scenario of estimates in Section 3.2 is the following:

- (i) We formalize tokens in the ring as independent random walks (Lemma 2).
- (ii) Noting the minimal meeting time of random walks of Item (i), we investigate correspondent one dimensional random walks (Lemma 3).

- (iii) To estimate hitting times of one dimensional random walks of Item (ii), we use the *coupling random walk* whose expected time can be gotten by some difference equations (Lemmas 4 and 5).
- (iv) Using the hitting time of Item (iii), we estimate the desirable expected time. Moreover we give a simple example satisfying Eqn. (9).

2 Herman's Algorithm

Let $L = \{0, \dots, n-1\} \simeq \mathbf{Z}/n\mathbf{Z}$ be a process ring, where n is an integer satisfying $n \geq 3$. Henceforth any index is defined for any integer subscript with modulo n . The state of each process is a single bit in $\{0, 1\}$, so that we denote by $\mathcal{X} = \{0, 1\}^n$ the configuration space whose element is represented by a vector of n bits $x = (x_0, \dots, x_{n-1}) \in \mathcal{X}$. For a given $0 < r < 1$ a random procedure $f : \mathcal{X} \rightarrow \mathcal{X}$ is defined by

$$f(x) = f_r(x) = y = (y_0, \dots, y_{n-1}) \in \mathcal{X}, \quad y_i = \begin{cases} x_{i-1} & \text{if } x_i \neq x_{i-1} \\ \gamma_i & \text{if } x_i = x_{i-1}, \end{cases} \quad (1)$$

where $\gamma_i, (i = 1, \dots, n)$ are independent random variables with each distribution

$$\Pr(\gamma_i = x_i) = 1 - \Pr(\gamma_i = 1 - x_i) = r \quad \text{for } i = 0, \dots, n-1. \quad (2)$$

For a given $x = (x_0, \dots, x_{n-1}) \in \mathcal{X}$, we define a local state to be any consecutive pair (x_{i-1}, x_i) :

- if $x_i = x_{i-1}$, then there is a *token* at process i . (We will also say that x_i is a token.)
- if $x_i \neq x_{i-1}$, then there is a *shift* at process i . (We will also say that x_i is a shift.)

Note that if n is odd, the number of tokens is also odd, that is,

$$\|x\| \bmod 2 \equiv n \bmod 2 \quad \text{for } x \in \mathcal{X}, \quad (3)$$

where $\|\cdot\|$ is the number of tokens in the ring. For each $x = (x_0, \dots, x_{n-1}) \in \mathcal{X}$ define the set of tokens/shifts as follows:

$$\begin{aligned} L_T &= L_T(x) = \{1 \leq i \leq n : x_i = x_{i-1}\}, \\ L_S &= L_S(x) = L \setminus L_T = \{1 \leq i \leq n : x_i \neq x_{i-1}\}. \end{aligned} \quad (4)$$

Moreover we also define $\mathcal{X}^k = \{x \in \mathcal{X} : |L_T(x)| = k\} = \{x \in \mathcal{X} : \|x\| = k\}$. We call the element of \mathcal{X}^1 a *legitimate* configuration. Use notation $f^t(x) = f(f^{t-1}(x))$ for $t \geq 1$, where $f^0(x) = x$. Herman proved the following theorem:

Theorem 1 (Herman). *For the random procedure f defined by Eqn. (1), the following statements hold:*

- (i) *The number of tokens does not increase:*

$$\Pr(\|x\| \geq \|f(x)\|) = 1. \quad (5)$$

(ii) If there exists a unique token at i , then the token is eventually at $i+1$: For each $x \in \mathcal{X}^1$ with $x_i = x_{i-1}$

$$\Pr(\exists k > 0 \text{ s.t. } y = f^k(x) \in \mathcal{X}^1, y_i = y_{i+1}) = 1. \quad (6)$$

(iii) If there are more than two or more tokens, the number of tokens is eventually strictly decreasing:

$$\Pr(\exists k > 0 \text{ s.t. } \|f^k(x)\| < \|x\|) = 1 \quad \text{for } \|x\| \geq 2. \quad (7)$$

Remark 1. (i) Suppose that n is odd. Then by Eqn. (3), the number of tokens is also odd. By Eqns. (5) and (7) we see that the number of decreasing tokens is even (including 0), and at last there will hereafter become a unique token.

(ii) By Eqn. (6), the algorithm progresses in a legitimate configuration, that is, the token circulates in the ring.

3 Expected Times for Herman's Algorithm

3.1 Main Results

Fix an odd integer $n \geq 3$. Then by Item (i) in Remark 1, we see that for $x \in \mathcal{X}$ there exists $t = t(x) < \infty$ with probability 1 such that

$$\|x\| \geq \|f(x)\| \geq \dots \geq \|f^{t-1}(x)\| > \|f^t(x)\| = \|f^{t+1}(x)\| = \dots = 1.$$

Now we analyze the above random time t for any $x \in \mathcal{X}$.

Theorem 2. Fix an odd integer $n \geq 3$. Then for any parameter $0 < r < 1$ of Eqn. (2), the expected time to become a legitimate configuration is less than $\frac{\pi^2 - 8}{8r(1-r)}n^2$, that is,

$$\max_{x \in \mathcal{X}} \mathbf{E} [\inf\{t \geq 0 : f^t(x) \in \mathcal{X}^1\}] < \frac{\pi^2 - 8}{8r(1-r)}n^2. \quad (8)$$

Moreover there exists $x \in \mathcal{X}$ such that

$$\mathbf{E}[\inf\{t \geq 0 : f^t(x) \in \mathcal{X}^1\}] = \Theta(n^2). \quad (9)$$

3.2 Proof of Theorem 2

For a fixed $3 \leq k \leq n$, we consider a configuration with k tokens $x^k \in \mathcal{X}^k$. Then we define the *first decreasing time* for $x^k \in \mathcal{X}^k$:

$$T(x^k) = \inf\{t > 0 : f^t(x^k) \notin \mathcal{X}^k\}. \quad (10)$$

To investigate the time, we define the minimum token distance: For $x^k \in \mathcal{X}^k$, remembering Eqn. (4),

$$D(x^k) = \begin{cases} \min\{d(i, j) : i \neq j \in L_T(x^k)\}, & \text{if } k \geq 2 \\ 0, & \text{if } k = 1 \end{cases},$$

where $d(i, j)$ denotes the minimum distance between i and j , that is,

$$d(i, j) = \min\{(i - j) \bmod n, (j - i) \bmod n\}. \quad (11)$$

Then it is easy to see that

$$D(x^k) \leq \lfloor n/k \rfloor \quad \text{for } x^k \in \mathcal{X}^k. \quad (12)$$

In fact if $D(x^k) > \lfloor n/k \rfloor$ then any distance of two tokens is greater than n/k then the sum of these distance is greater than n . Now the repetition of a random procedure f implies a stochastic chain $D(f^t(x^k))$ for $t = 0, 1, 2, \dots$ on $\{0, 1, \dots, \lfloor n/k \rfloor\}$. Lemma 3 will claim the statistical property of $D(f(x^k))$. Defining

$$\tau_k = \max_{x^k \in \mathcal{X}^k} \mathbf{E}[T(x^k)], \quad (13)$$

we investigate the expected time that configuration x^k will be legitimate, which is defined by

$$\mathbf{T}_k = \max_{x^k \in \mathcal{X}^k} \mathbf{E}[\inf\{t \geq 0 : f^t(x^k) \in \mathcal{X}^1\}] = \max_{x^k \in \mathcal{X}^k} \mathbf{E}[\inf\{t \geq 0 : D(f^t(x^k)) = 0\}].$$

Noting Item (i) in Remark 1, we see that \mathbf{T}_k can be bounded by $\{\tau_k\}_{k=1}^n$ as follows:

Lemma 1. *For an odd integer $n \geq 3$, assume that there exist $3 \leq k \leq n$ tokens in an n -ring. Then we have*

$$\mathbf{T}_k \leq \sum_{l=1}^{(k-1)/2} \tau_{2l+1}. \quad (14)$$

Proof. Since n is odd, k is also odd by Theorem 1. Now we suppose that the number of tokens will be k_1 at the first decreasing time $T(x^k)$. Putting $k = k_0$, similarly suppose that the number of k_i tokens will be k_{i+1} at $T(x^{k_i})$ for $i = 0, \dots, l$, and $k_{l+1} = 1$. Note that $k = k_0 > k_1 > \dots > k_l > k_{l+1} = 1$ are well-defined by Item (i) in Remark 1. It is clear that k_i depends on x^k for $i = 1, \dots, l$. By the definition of \mathbf{T}_k , we have $\mathbf{T}_k = \max_{x^k \in \mathcal{X}^k} \mathbf{E}[T(x^{k_0}) + T(x^{k_1}) + \dots + T(x^{k_l})]$. Since k_i is odd for $i = 0, 1, \dots, l$ and $k_l \geq 3$, we obtain

$$\begin{aligned} \mathbf{T}_k &\leq \max_{x^k \in \mathcal{X}^k} \mathbf{E}[T(x^k)] + \max_{x^{k-2} \in \mathcal{X}^{k-2}} \mathbf{E}[T(x^{k-2})] + \dots + \max_{x^3 \in \mathcal{X}^3} \mathbf{E}[T(x^3)] \\ &= \tau_k + \tau_{k-2} + \dots + \tau_3. \quad \square \end{aligned}$$

Computing local probability of the position of each token, we will estimate τ_i for $i = 3, 5, \dots, k$. The movement of each token in the ring is characterized by the following random walks:

Lemma 2. *For an odd integer $n \geq 3$, assume that there exist $3 \leq k \leq n$ tokens in an n -ring. Putting each position of k tokens of x^k as*

$$S_0^1, \dots, S_0^k \in L_T(x^k) \text{ sequentially,} \quad (15)$$

consider independent k random walks S_t^1, \dots, S_t^k defined by Eqn. (15) and

$$\Pr(S_t^l = i + 1 | S_{t-1}^l = i) = 1 - \Pr(S_t^l = i | S_{t-1}^l = i) = r \quad \text{for } l = 1, \dots, k, \quad 1 \leq t \leq t^*, \quad (16)$$

where t^* is the first coalescing time of $\{S_t^l\}_{l=1}^k$, that is,

$$t^* = t^*(x^k) = \inf\{t > 0 : 1 \leq \exists i < \exists j \leq k \text{ s.t. } S_t^i = S_t^j\}. \quad (17)$$

Then the process of the position of tokens is equivalent to the process of S_t^1, \dots, S_t^k for $0 \leq t \leq t^* - 1$. Moreover the distribution for t^* is equivalent to the one of $T(x^k)$ defined by Eqn. (10).

Proof. For $x^k = x = (x_0, \dots, x_{n-1}) \in \mathcal{X}^k$, we will show this lemma with either $D(x) \geq 2$ or $D(x) = 1$.

Assume that $D(x) \geq 2$, that is, if $i \in L_T(x)$ then $i - 1, i + 1 \in L_S(x)$. By the definition of the shift/token, we have that $x_{i-1} = x_i$, $x_{i-2} \neq x_{i-1}$ and $x_i \neq x_{i+1}$. Therefore we obtain that

$$\begin{aligned} & \Pr(\{i \in L_S(f(x))\} \cap \{i + 1 \in L_T(f(x))\}) \\ &= 1 - \Pr(\{i \in L_T(f(x))\} \cap \{i + 1 \in L_S(f(x))\}) = r \end{aligned}$$

by Eqns. (1) and (2). Hence we see that

$$\Pr(\|f(x)\| = \|x\|) = 1 \quad \text{for } D(x) \geq 2, \quad (18)$$

and the local transition probability of the position of the token is equivalent to Eqn (16) for $D(x) \geq 2$.

Next suppose that $D(x) = 1$, that is, there exists $2 \leq m \leq k$ satisfying that $i, i + 1, \dots, i + m - 1 \in L_T(x), i - 1, i + m \in L_S(x)$. Then we see that $x_{i-1} = x_i = x_{i+1} = \dots = x_{i+m-1}$ and $x_{i-2} \neq x_{i-1}, x_{i+m-1} \neq x_{i+m}$. Putting $y = (y_0, \dots, y_{n-1}) = f(x)$, we have that $y_{i-1} \neq y_{i+m}$. On the other hand, assume that continual m particles at time $t - 1$, that is, $S_{t-1}^1 = i, S_{t-1}^2 = i + 1, \dots, S_{t-1}^m = i + m - 1$, so that we regard these particles as tokens. At the next step, consider that there is a token (resp. shift) if the number of particles of $\{S_t^l\}_{l=1, \dots, m}$ on each position $i \leq j \leq i + m$ is one (resp. zero or two). Noting that y_j is a token if $y_{j-1} = y_j$ for $j = i, \dots, i + m$, we have that the distribution of process for continual m are equivalent to the distribution of S_t^1, \dots, S_t^m for $0 \leq t \leq t^* - 1$. It is clear that the equivalence of the law of $T(x)$ and the first collision time $t^*(x)$. \square

By Lemma 2 we see that the process of each token is simulated by the random walk defined by Eqn. (16). We will give some estimate of the transition probability for $D(f(x))$ as the following lemma:

Lemma 3. *For an odd integer $n \geq 3$, assume that there exist $3 \leq k \leq n$ tokens in an n -ring. Then the following statements hold:*

- (i) *If a configuration $x \in \mathcal{X}^k$ satisfies $D(x) = 1$ then there exist p_x, q_x such that $p_x \leq r(1 - r) \leq q_x < 1 - p_x$ and*

$$\begin{aligned} \Pr(\{D(f(x)) = 2\} \cap \{\|f(x)\| = k\}) &= p_x, \quad \Pr(\|f(x)\| < k) = q_x, \\ \Pr(\{D(f(x)) = 1\} \cap \{\|f(x)\| = k\}) &= 1 - p_x - q_x. \end{aligned}$$

- (ii) If a configuration $x \in \mathcal{X}^k$ satisfies $D(x) \geq 2$, then $f(x) \in \mathcal{X}^k$.
 (i) If $2 \leq D(x) \leq \lfloor n/k \rfloor - 1$ then there exist p_x, q_x such that $p_x \leq r(1-r) \leq q_x < 1 - p_x$ and

$$\begin{aligned} \Pr(D(f(x)) = D(x) + 1) &= p_x, \quad \Pr(D(f(x)) = D(x) - 1) = q_x, \\ \Pr(D(f(x)) = D(x)) &= 1 - p_x - q_x. \end{aligned}$$

- (ii) If $D(x) = \lfloor n/k \rfloor \geq 2$ then there exists $r(1-r) \leq q_x < 1$ such that

$$\Pr(D(f(x)) = \lfloor n/k \rfloor - 1) = q_x, \quad \Pr(D(f(x)) = \lfloor n/k \rfloor) = 1 - q_x.$$

The rough sketch of estimating transition probabilities in Lemma 3 is in Fig. 1.

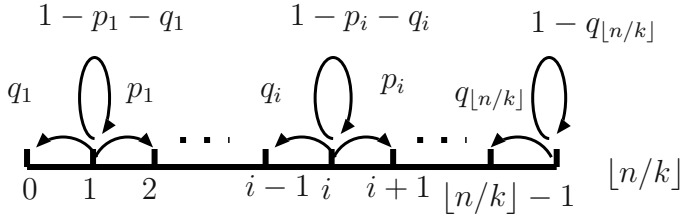


Fig. 1. Illustration of the transition probabilities of $D(f^t(x))$ in Lemma 3. Each suffix of the axis denotes not $f^t(x) \in \mathcal{X}$ but $D(f^t(x)) \in \{0, 1, \dots, \lfloor n/k \rfloor\}$. Therefore for example the probability p_i presents p_x in Lemma 3 for some $D(x) = i$. Hence we see that $0 < p_i \leq r(1-r) \leq q_i < 1$ for $i = 1, \dots, \lfloor n/k \rfloor$.

Proof of Lemma 3. Fix $x \in \mathcal{X}^k$. By Lemma 2, we can regard the process of k tokens as k independent random walks defined by Eqns. (15) and (16). For convenience, we define $L_T^{\min} = L_T^{\min}(x) = \{i \in L_T(x) : i, i + D(x) \in L_T(x)\}$, which is not empty. Noting that S_0^1, \dots, S_0^k are defined by Eqn. (15), we have that

$$\Pr(d(S_0^i, S_0^{i+1}) = D(x)) = 1 \quad \text{for any } i \in L_T^{\min}. \quad (19)$$

By Eqn. (16), we see that

$$\begin{aligned} \Pr(d(S_1^i, S_1^{i+1}) = D(x) - 1) &= r(1-r), & \Pr(d(S_1^i, S_1^{i+1}) = D(x)) &= r^2 + (1-r)^2, \\ \Pr(d(S_1^i, S_1^{i+1}) = D(x) + 1) &= r(1-r) & \text{for any } i \in L_T^{\min}. \end{aligned} \quad (20)$$

- (i) If $D(x) = 1$, then using Eqn. (20) we obtain that

$$\begin{aligned} p_x &= \Pr(\{D(f(x)) = 2\} \cap \{\|f(x)\| = \|x\|\}) \\ &\leq \Pr\left(\bigcap_{i \in L_T^{\min}(x)} \{d(S_1^i, S_1^{i+1}) = 2\}\right) \leq r(1-r). \end{aligned} \quad (21)$$

Moreover we also have that

$$1 > q_x = \Pr(\|f(x)\| < k) = \Pr\left(\bigcup_{i \in L_T^{\min}(x)} \{d(S_1^i, S_1^{i+1}) = 0\}\right) \geq r(1 - r). \quad (22)$$

The first inequality of Eqn. (22) holds, because $\Pr(\|f(x)\| = k) \geq \Pr(f(x) = x) > 0$. Hence Item (i) has been proved.

(ii) Since $D(x) \geq 2$, Eqn. (18) holds. Using Lemma 2,

$$\Pr(D(f(x)) = D(x)) + \Pr(D(f(x)) = D(x) + 1) + \Pr(D(f(x)) = D(x) - 1) = 1. \quad (23)$$

(i) $2 \leq D(x) \leq \lfloor n/k \rfloor - 1$: Then we can prove that

$$\Pr(D(f(x)) = D(x) + 1) \leq r(1 - r) \leq \Pr(D(f(x)) = D(x) - 1) < 1, \quad (24)$$

by the same method of Eqns (21) and (22). Hence by Eqn. (23), Item (a) has been proved.

(ii) $D(x) = \lfloor n/k \rfloor$: Using the same method of Eqn. (22), we see that

$$\Pr(D(f(x)) = \lfloor n/k \rfloor - 1) \geq r(1 - r).$$

Since $D(x) \leq \lfloor n/k \rfloor$ for $x \in \mathcal{X}$, we have that $\Pr(D(f(x)) = \lfloor n/k \rfloor + 1) = 0$. Hence by Eqn. (23), Item (b) has been proved. \square

Remark 2. There exist an n -ring and a configuration $x \in \mathcal{X}$ satisfying that equality holds in Eqns (21), (22) and (24) respectively.

We will compare $D(f^t(x^k))$ to the following random walks:

Lemma 4. Consider the following Markov chain X_t on state space $\{0, 1, \dots, m\}$ with absorbing barrier 0: For $t \geq 0$ and $i = 1, \dots, m - 1$

$$\begin{aligned} \Pr(X_{t+1} = i | X_t = i) &= r^2 + (1 - r)^2, \\ \Pr(X_{t+1} = i + 1 | X_t = i) &= \Pr(X_{t+1} = i - 1 | X_t = i) = r(1 - r) \end{aligned} \quad (25)$$

and for $t \geq 0$

$$\begin{aligned} \Pr(X_{t+1} = 0 | X_t = 0) &= 1, \quad \Pr(X_{t+1} = m | X_t = m) = 1 - r(1 - r), \\ \Pr(X_{t+1} = m - 1 | X_t = m) &= r(1 - r). \end{aligned} \quad (26)$$

Then the expected hitting time of the absorbing barrier 0 is

$$\max_{1 \leq i \leq m} \mathbf{E}[\inf\{t \geq 0 : X_t = 0\} | X_0 = i] = \mathbf{E}[\inf\{t \geq 0 : X_t = 0\} | X_0 = m] = \frac{m(m+1)}{2r(1-r)}.$$

Proof. Putting $a_j = \mathbf{E}[\inf\{t \geq 0 : X_t = 0\} | X_0 = j]$, we have that

$$\begin{cases} a_{j+1} - 2a_j + a_{j-1} = -\frac{1}{r(1-r)} & \text{for } j = 1, \dots, m-1 \\ a_0 = 0, a_m = a_{m-1} + \frac{1}{r(1-r)}. \end{cases}$$

Hence $a_j = a_{j-1} + \frac{m-j+1}{r(1-r)}$ for $j = 1, \dots, m$. Noting $a_0 = 0$, sum up the equations, so that we obtain $a_m = \frac{m(m+1)}{2r(1-r)}$. \square

To estimate τ_k , we use the coupling chain X_t in Lemma 4 which is defined in the same probability space for $D(f^t(x^k))$.

Lemma 5. *For an odd integer $n \geq 3$, assume that there exist k tokens for $3 \leq k \leq n$ in an n -ring. Then we have $\tau_k \leq \frac{n^2}{r(1-r)k^2}$.*

Proof. Putting $m = \lfloor n/k \rfloor$ in Lemma 4, we define the stochastic process X_t satisfying $X_0 = D(f^0(x^k))$ and Eqns. (25), (26), which is defined on the state space $\{0, \dots, \lfloor n/k \rfloor\}$. Then we see that $D(f^t(x^k))$ is stochastically smaller than X_t , that is, $\Pr(D(f^t(x^k)) \leq X_t) = 1$ (see [12, Chapter IV]). Therefore putting $T_X = \inf\{t > 0 : X_t = 0\}$, since $\lfloor n/k \rfloor \geq 1$,

$$\tau_k \leq \mathbf{E}[T_X | X_0 = \lfloor n/k \rfloor] = \frac{\lfloor n/k \rfloor(\lfloor n/k \rfloor + 1)}{2r(1-r)} \leq \frac{1}{r(1-r)} \left(\frac{n}{k}\right)^2. \quad \square$$

Proof of Theorem 2. By Eqn. (14) and Lemma 5

$$T_k \leq \frac{n^2}{r(1-r)} \left(\frac{1}{3^2} + \frac{1}{5^2} + \dots + \frac{1}{k^2} \right) < \frac{\pi^2 - 8}{8r(1-r)} n^2 \quad \text{for any } k = 3, 5, \dots, n. \quad (27)$$

In fact the last inequality of Eqn. (27) holds because it is known that

$$\sum_{j=0}^{\infty} \frac{1}{(2j+1)^2} = \frac{\pi^2}{8}$$

(see e.g. [10]). On the other hand, for a sufficiently large n , consider the token $x = (x_0, \dots, x_{n-1}) \in \mathcal{X}^3$ with $x_0, x_{\lfloor n/6 \rfloor}, x_{\lfloor n/2 \rfloor} \in L_T(x)$. Now set random walks S_t^1, S_t^2, S_t^3 with initial positions $S_0^1 = x_0$, $S_0^2 = x_{\lfloor n/6 \rfloor}$, $S_0^3 = x_{\lfloor n/2 \rfloor}$, and transition probabilities satisfying Eqn. (16) respectively. We see that $D(x) = d(S_0^1, S_0^2) = \lfloor n/6 \rfloor$. Moreover $D(f^t(x)) = d(S_t^1, S_t^2)$ for $t \in \{t : d(S_t^1, S_t^2) \leq \lfloor n/3 \rfloor\}$. Until the time, we have

$$\begin{aligned} \Pr(D(f^t(x)) = D(f^{t-1}(x)) + 1) &= \Pr(D(f^t(x)) = D(f^{t-1}(x)) - 1) = r(1-r), \\ \Pr(D(f^t(x)) = D(f^{t-1}(x))) &= r^2 + (1-r)^2. \end{aligned}$$

Remembering $T(x)$ as Eqn. (10), we have $\mathbf{E}[T(x)] \geq \mathbf{E}[T_* | D(x) = \lfloor n/6 \rfloor]$, where $T_* = \inf\{t > 0 : D(f^t(x)) \in \{0, \lfloor n/3 \rfloor\}\}$. Now putting $b_i = \mathbf{E}[T_* | D(x) = i]$, ($i = 0, \dots, \lfloor n/3 \rfloor$), the following difference equation holds: $b_i = \{r^2 + (1-r)^2\}b_i + r(1-r)b_{i+1} + r(1-r)b_{i-1} + 1$ for $i = 1, \dots, \lfloor n/3 \rfloor - 1$ with boundary conditions $b_0 = b_{\lfloor n/3 \rfloor} = 0$. It is easy to get the solution $b_i = i(\lfloor n/3 \rfloor - i)/\{2r(1-r)\}$ for $i = 0, \dots, \lfloor n/3 \rfloor$. Therefore we see that $b_{\lfloor n/6 \rfloor} \geq n^2/\{100r(1-r)\}$, so that $T_3 \geq \mathbf{E}[T(x)] \geq n^2/\{100r(1-r)\}$. Hence we have Eqn. (9). \square

Remark 3. By the proof of Eqn. (9), we see that a few tokens with long distances make the expected time large.

4 Concluding Remarks

We proved that the maximal expected time of Herman's self-stabilizing algorithm is less than $\frac{\pi^2-8}{8r(1-r)}n^2$. Especially if $r = 1/2$ it is bounded by $0.936n^2$. However we do not know that whether $r = 1/2$ is optimal, which is a future work. It would also be of future interest to apply our argument to general connected graphs.

Moreover note that recently Fribourg et. al. [7] characterized Herman's algorithm with Gibbs fields. We should study the self-stabilization at the viewpoint of statistical physics.

References

1. Adler, Ahn, Karp and Ross, Coalescing Times for IID Random Variables with Applications to Population Biology, *Random Struct. Alg.*, **23**, 2, 155–166, (2003).
2. Aldous and Fill, *Reversible Markov Chains and Random Walk on Graphs*, monograph in preparation,
<http://www.stat.berkeley.edu/~aldous/book.html>
3. Brightwell and Winkler, Maximal hitting time for random walks on graphs, *Random Struct. Alg.*, **1**, 263–275, (1990).
4. D. Coppersmith, P. Tetali and P. Winkler, Collisions among random walks on a graph, *SIAM J. Disc. Math.* **6**, 3, (1993), 363–374.
5. E. Dijkstra, Self-stabilizing system in spite of distributed control, *Comm. ACM*, **17** (11), 643–644, Nov. 1974.
6. Dolev, *Self-Stabilization*, MIT Press, 2000.
7. L. Fribourg, S. Messika and C. Pecaronny, Trace of Randomized distributed algorithms as Gibbs fields, *Research Report LSV-02-12*, Sep. 2002, Lab. Spéc. et. Vér, CNRS & ENS de Cachan, France.
8. Herman, Probabilistic self-stabilization, *Info. Proc. Lett.*, **35**, (1990), 63–67.
9. Y. Hassin and D. Peleg, Distributed Probabilistic Polling and Applications to Proportionate Agreement, *Proc. of ICALP99, LNCS 1644*, 402–411, (1999).
10. Hofbauser, A simple proof of $1 + \frac{1}{2^2} + \frac{1}{3^2} + \dots = \frac{\pi^2}{6}$ and related identities, *American Mathematical Monthly*, **109**, (2002), 196–200.
11. Lamport, Solved problems, unsolved problems and non-problems in concurrency, *Proc. of PDOC*, 1–11, 1984.
12. Lindvall, *Lectures on the coupling method*, Wiley, 1992.
13. Tetali and Winkler, Simultaneous reversible Markov chains, *Combinatorics: Paul Erdős is Eighty*, vol. 1, 433–451, Bolai Soc. Math. Stud. (1993).

An Efficient Online Algorithm for Square Detection

Ho-Fung Leung¹, Zeshan Peng², and Hing-Fung Ting²

¹ Department of Computer Science and Engineering,
The Chinese University of Hong Kong, Hong Kong
`lhf@cse.cuhk.edu.hk`

² Department of Computer Science, The University of Hong Kong, Hong Kong
`{zspeng,hfting}@csis.hku.hk`

Abstract. A square is a string of characters that can be divided into two identical substrings. The problem of detecting squares in a string finds applications in many areas such as bioinformatics and data compression. In this paper, we give the first efficient online algorithm for the problem. Given any input string, our algorithm reports the first square in the string using $O(n \log^2 n)$ time where n is the position in the string where the square ends. This time complexity is only a factor of $O(\log n)$ larger than that of an optimal offline algorithm.

1 Introduction

Given a string $T = \langle T[1], T[2], \dots, T[2k] \rangle$ of an even number of characters, we say that T is a *square* if we can divide T into two identical substrings, i.e., if $\langle T[1], T[2], \dots, T[k] \rangle = \langle T[k+1], T[k+2], \dots, T[2k] \rangle$. We say that a string $T = \langle T[1], T[2], \dots, T[n] \rangle$ contains a square if it has a substring $\langle T[i], T[i+1], \dots, T[j] \rangle$ which is a square. It is *square free* if it has no such substrings. Detecting squares has found applications in areas such as bioinformatics [5] and text compression [4]. There are many efficient offline algorithms for the problem [1–3, 8, 10]. However, this offline property is not desirable in many applications. For example, suppose we want to determine whether a string of one million characters is square free. Note that we can stop as soon as we detect the first square. However, if we apply the offline algorithms, we have to scan through all the one million characters even when the square appears at the first few characters of the string. In some applications, such as online data compression, this offline property is even unacceptable; we need to detect a square as soon as a new character arrives.

Our work on constraints satisfaction gives us another motivation for studying online algorithms for square detection. In [6, 7, 11], we studied the local search method for solving the classical constraints satisfaction problem. This method is efficient in general. However, there is no guarantee its stop; the search can be trapped in some infinite loop. Interestingly, it can be proved that such a loop corresponds to a square in some string whose characters encode the states of the search. To avoid wasting time in an infinite loop, we need to detect a square

in the string online. In other words, every time the search makes a move and a new character is added to the string, we detect if there is any square in the new string. We stop if there is a square.

In this paper, we give the first online algorithm for square detection. Given any input string T , our algorithm reads T character by character. After every new character is read, it checks immediately whether there is a square in the string. If the first square ends at the n th position of T , our algorithm stops after reading the first n characters and report the square. The time complexity of the whole process is $O(n \log^2 n)$. In other words, after reading a character, it takes $O(\log^2 n)$ amortized time for square detection. The optimal offline algorithm for the problem [3] has time complexity $O(n \log n)$ for a *general* alphabet where n is the length of T . Note that if the size of the alphabet is constant, offline deciding whether a string is square free can be solved in $O(n)$ [9] time.

To explain how our algorithm works, let us suppose that we have read a string $T = \langle T[1], T[2], \dots, T[n] \rangle$ of n characters and find that it is square free. We have to continue. A new character $T[n+1]$ arrives. Although for each $1 \leq i \leq n$, the string $\langle T[i], T[i+1], \dots, T[n] \rangle$ is not a square (otherwise we would have stopped), it is possible that for some i , $\langle T[i], T[i+1], \dots, T[n], T[n+1] \rangle$ is a square. To check these n strings $\langle T[i], T[i+1] \dots T[n+1] \rangle$, $1 \leq i \leq n$, all over again takes too much time. To reduce the time complexity, our idea is to identify $O(\log n)$ regions in the string. We show that if any one of these regions has a square, it will have some structural property. By using this property, we show how to recover the square in $O(\log n)$. Since a square can start at anyone of the $O(\log n)$ regions, we have to check them all, and the total time for such a detection is $O(\log^2 n)$.

The paper is organized as follows. In Section 2, we give the definitions and notations necessary for our discussion. In Section 3, we give a framework of our algorithm. In Section 4, we give details of the main step and analyze its running time. Finally, we propose some future work in Section 5.

2 Definitions and Notations

In this section, we give formally the definitions and notations necessary for our discussion.

The input of our problem is a string $T = \langle T[1], T[2], T[3], \dots \rangle$ of characters. Given any integers $1 \leq a \leq b$, let $T[a..b]$ denote the string $\langle T[a], T[a+1], \dots, T[b] \rangle$. A string $S = T[a..b]$ is a *square* if

- its length $\ell = b - a + 1$ is even, and
- $T[a..(a + \ell/2 - 1)] = T[(a + \ell/2)..b]$, i.e., $T[a] = T[a + \ell/2], T[a + 1] = T[a + \ell/2 + 1], \dots, T[a + \ell/2 - 1] = T[b]$.

We say that $T[a..(a + \ell/2 - 1)]$ is the *first half*, and $T[(a + \ell/2)..b]$ is the *second half*, of S . Given another string $T[i..j]$, we say that S is *hanging in* $T[i..j]$ if the first half of S falls completely in $T[i..j]$ and the second half of S does not fall completely in $T[i..j]$. More precisely, we say that S is hanging in $T[i..j]$ if we have

$$i \leq a \leq a + \frac{\ell}{2} - 1 \leq j < b. \quad (1)$$

For ease of referencing, we also say that S is a *hanging square* in $T[i..j]$.

For any $h \geq 1$, a *suffix* of $T[1..h]$ is the string $T[i..h]$ for some $1 \leq i \leq h$. We call $T[i..h]$ a *square suffix* of $T[1..h]$ if it is also a square. For any two strings X and Y , let XY denote the concatenation of X and Y , i.e., the string obtained by appending Y at the end of X . Hence, a square is the string XX for some X .

3 The Algorithm

In this section, we describe an online algorithm for detecting squares. Our algorithm reads the input string T character by character. After reading every new character $T[h]$, it checks immediately (i.e., without reading $T[h+1]$) whether the string $\langle T[1], T[2], \dots, T[h] \rangle$ has a square. If there is a square, it stops and reports the square. Note that if h is the smallest integer such that $T[1..h]$ has a square, then this square must be a suffix of $T[1..h]$. Furthermore, $T[1..(h-1)]$ must be square free. Thus, we focus on detecting a square suffix of $T[1..h]$, and our analysis will make use of the fact that $T[1..(h-1)]$ is square free.

The core of our algorithm is the procedure $\text{DHSq}(i, j)$, which solves the following more restrictive problem: For every $j < h \leq 2j - i + 1$, after the character $T[h]$ is read, $\text{DHSq}(T[i..j])$ checks whether $T[1..h]$ has a square suffix hanging in $T[i..j]$.

Note that for all $h > 2j - i + 1$, there is no square suffix of $T[1..h]$ that can hang in $T[i..j]$. In Section 4, we describe the procedure $\text{DHSq}(i, j)$ and show that it can be finished in total $O((j-i) \log(j-i))$ time. In the following, we explain how to make use of DHSq to solve the online square detection problem.

Our idea is to run enough instances of $\text{DHSq}(i, j)$ for different i, j such that if $T[1..h]$ has a square suffix S , then there must be an instance $\text{DHSq}(i, j)$ running where S is a hanging square of $T[i..j]$. Then, when $T[h]$ is read, $\text{DHSq}(i, j)$ will detect S . Here are the details.

For any $1 \leq i \leq j$, we say that the pair (i, j) is a level- ℓ pair if

- the second integer $j = k \cdot 2^\ell$ for some $k \in \{1, 2, 3, \dots\}$, and
- the first integer $i = \max\{1, k2^\ell - 4 \cdot 2^\ell + 1\}$.

Following is our online square detection algorithm DSq , whose task is basically to start the right $\text{DHSq}(i, j)$ at the right time.

For $h = 1, 2, 3, \dots$, after reading $T[h]$, do the following steps:
if there is a square in $T[(h-3)..h]$, or any of the running $\text{DHSq}(i, j)$ detects a square in $T[1..h]$, then stops.
 $j = h$; $\ell = 0$;
while ($j \geq 2^\ell$) **do**
 if ($j = k2^\ell$ for some k)
 $i = \max\{1, k2^\ell - 4 \cdot 2^\ell + 1\}$;
 for the level- ℓ pair (i, j) , start $\text{DHSq}(i, j)$;
 $\ell = \ell + 1$;

(Note that **Dsq** is not a parallel algorithm. After reading some character $T[h]$, each $\text{DHangSq}(i, j)$ will check in turn whether $T[1..h]$ has a square suffix hanging in $T[i..j]$.) Now, we prove the correctness of **Dsq**. Obviously, **Dsq** correctly detects a square suffix XX if $|XX| \leq 4$. The following lemma considers the other case.

Lemma 1. *Suppose h is the smallest integer such that $T[1..h]$ contains a square suffix XX . Furthermore, suppose that $|XX| > 4$. Then, after $T[h]$ is read, the square XX will be detected by one of the $\text{DHangSq}(i, j)$ started by **Dsq**.*

Proof. Since $|XX| > 4$, we have $|X| > 2$. Suppose that $2^\ell < |X| \leq 2^{\ell+1}$ for some $\ell \geq 1$. Note that there exists integer k such that $k2^\ell < h \leq (k+1)2^\ell$. Let $j = k2^\ell$ and $i = k2^\ell - 4 \cdot 2^\ell + 1$. Hence, (i, j) is a level- ℓ pair, and by construction, when the character $T[j]$ is read, **Dsq** will run $\text{DHangSq}(i, j)$. Below, we verify that XX must be hanging in $T[i..j]$, and thus it will be detected by $\text{DHangSq}(i, j)$ (see Theorem 4).

Note that (i) since $|X| > 2^\ell$ and $h \leq (k+1)2^\ell$, we have $h - |X| < k2^\ell = j$ and thus the first half of the square ends at some position smaller than j , and (ii) we have $i = j - 4 \cdot 2^\ell + 1 < h - 4 \cdot 2^\ell + 1 \leq h - 2|X| + 1$ and thus the starting position of XX is not smaller than i . Therefore, XX hangs in $T[i..j]$. \square

Finally, we have the following theorem.

Theorem 1. *Suppose h is the smallest integer such that $T[1..h]$ contains a square. Then, for every $1 \leq i \leq h$, **Dsq** correctly determines whether $T[1..i]$ has a square suffix immediately after reading $T[i]$. The total time complexity is $O(h \log^2 h)$.*

Proof. The correctness of **Dsq** is given in Lemma 1. To show the time bound, it suffices to show below that all instances $\text{DHangSq}(i, j)$ created by **Dsq** can be finished in total $O(h \log^2 h)$ time.

Note that **Dsq** stops before reading the character $T[2^{\lceil \log h \rceil}]$. Hence, it will not run $\text{DHangSq}(i, j)$ for any level- $\lceil \log h \rceil$ pair (i, j) . For any $1 \leq \ell < \lceil \log h \rceil$, **Dsq** runs DHangSq for $O(h/2^\ell)$ level- ℓ pairs. In Section 4, we show that each of these instances can be solved in $O(\ell 2^\ell)$ time (Theorem 4), and thus all these level- ℓ instances can be solved in total $O(\ell 2^\ell (h/2^\ell)) = O(\ell h)$ time. Summing this total running time over all the $O(\log h)$ different levels ℓ , the theorem follows. \square

4 Design and Analysis of DHangSq

In this section, we describe how to implement $\text{DHangSq}(i, j)$. First, we make the following observation.

Fact 2 *Suppose that the square $T[k..h]$ hangs in $T[i..j]$. Then, there are strings X and G such that $T[k..j] = XGX$ and $T[(j+1)..h] = G$.*

The above fact can be verified easily by an example. Note that the square $T[5..10] = abcabc$ hangs in $T[1..8] = aaaaabca$, and $T[5..8] = XGX$ and $T[9..10] = G$ where $X = a$ and $G = bc$. Fact 2 motivates the following definition: We say

that a string is a *pseudo-square* with a *center* G if it is equal to XGX for some X . For example, the string $abccab$ is a pseudo-square with a center cc . Followings are some simple facts about a pseudo-square.

- A pseudo-square with an empty string as its center is a square.
- A pseudo-square may have different centers. For example, the string $aabbbbbaa$ is a pseudo-square with a center $bbbb$, and is also a pseudo-square with a center $abbbba$.
- Every string is a pseudo-square with the center equal to itself.

Now, we replace Fact 2 in terms of pseudo-squares.

Fact 3 *Suppose $i \leq j \leq h$. The string $T[1..h]$ has a square suffix hanging in $T[i..j]$ if and only if $T[i..j]$ has a suffix that is a pseudo-square with a center equal to $T[(j+1)..h]$.*

The above fact suggests a simple way to implement $\text{DHangSq}(i, j)$. After reading $T[i], T[i+1], \dots, T[j]$, we find out the centers of all pseudo-square suffixes of $T[i..j]$ and put them in a list L . Then, after reading any $T[h]$ where $j < h \leq 2j - i + 1$, we conclude that $T[1..h]$ has a square suffix hanging in $T[i..j]$ if the string $T[(j+1)..h]$ is in L .

Note that L can have as many as $O((j-i)^2)$ entries. In the next section, we describe a much more compact list that is still sufficient. We also show how to find this list in $O(j-i)$ time. Then, in Section 4.2, we explain how to make use of this list to run $\text{DHangSq}(i, j)$ in $O((j-i) \log(j-i))$ time.

4.1 Minimum Suffix-Centers

In this section, we explain how to find a short list of centers for $\text{DHangSq}(i, j)$. We say G is a *suffix-center* (for $T[i..j]$) at position s ($i \leq s \leq j$) if $T[i..j]$ has a pseudo-square suffix $T[k..j] = XGX$ where G starts at $T[s]$. If G is the shortest among all the suffix-centers at s , we say that G is the *minimum suffix-center* at s . For example, if $T[3..10] = ccaabbaa$, bb is a suffix-center at 7, and obviously, it is minimum. The following lemma shows why we are interested in minimum suffix-centers.

Lemma 2. *Suppose that h is the smallest integer such that $T[i..h]$ contains a square suffix hanging in $T[i..j]$. Then, $G = T[(j+1)..h]$ is a minimum suffix-center at some $s \in \{i, i+1, \dots, j\}$.*

Proof. From Fact 3, we know that G is a suffix-center at some position s . To show that it is minimum, we assume to the contrary that another $G' \neq G$ is the minimum suffix-center at s . Note that both G and G' are substring of $T[i..j]$ starting at s , and because of G' is minimum, we have $|G'| < |G|$. It follows that G' is a prefix of G . Together with the fact that $T[(j+1)..h] = G$, we conclude that $T[(j+1)..h'] = G'$ for some $h' < h$. By Fact 3, $T[i..h']$ contains a square suffix hanging in $T[i..j]$. Hence, $T[h]$ is not the first character arrival such that $T[i..h]$ contains a square suffix hanging in $T[i..j]$. This is a contradiction. \square

DHangSq(i, j) is defined as follows:

1. Construct the list L of all minimum suffix-centers for $T[i..j]$;
2. For every $h \in \{j+1, j+2, \dots, 2j-i+1\}$, after reading $T[h]$, check whether $T[(j+1)..h]$ is in L . If $T[(j+1)..h]$ is in L , then stop and report finding a hanging square.

Below, we show how to construct L in $O(j-i)$ time (Lemma 4). In the next section, we show how to check, for each h , whether $T[(j+1)..h]$ is in L in $O(\log(j-i))$ time (Lemma 5). Altogether, we have the following theorem.

Theorem 4. *For every $h \in \{j+1, j+2, \dots, 2j-i+1\}$, after reading $T[h]$, DHangSq(i, j) correctly detects whether $T[i..h]$ contains a square suffix that hangs in $T[i..j]$. The total time complexity is $O((j-i) \log(j-i))$.*

Since there is at most one minimum suffix-center for each position $h \in \{i, i+1, \dots, j\}$, the list L of all minimum suffix-centers for $T[i..j]$ has size $O(j-i)$. Below, we show how to construct L in $O(j-i)$ time. Note that even though there are only $O(j-i)$ strings in L , many of them may have $\Theta(j-i)$ characters. Thus, we cannot afford to store L as a list of strings. Instead, we will only store, for each string in L , a pair of indices (s, e) where s is the starting position, and e is the ending position of this string.

The following lemma characterizes a minimum suffix-center.

Lemma 3. *Let G be the minimum suffix-center for $T[i..j]$ at s . Suppose G is the center of the pseudo-square suffix $T[k..j] = XGX$. Then, X is the longest suffix of $T[i..(s-1)]$ that is also equal to a suffix of $T[i..j]$.*

Proof. Since $T[k..j] = XGX$ and G starts at s , we have $T[k..(s-1)] = X$, $T[s..(s+|G|-1)] = G$, and $T[(s+|G|)..j] = X$. Hence, $T[k..(s-1)]$ is a suffix of $T[i..(s-1)]$ that is equal to a suffix of $T[i..j]$, namely $T[(s+|G|)..j]$. $T[k..(s-1)]$ is the longest such suffix because $|G|$ is minimum. \square

Thus, to find L , it suffices to find for all $i \leq s \leq j$, the longest suffix of $T[i..(s-1)]$ that is also a suffix of $T[i..j]$. The following lemma shows that they can be found efficiently.

Lemma 4. *We can find for all $i \leq k \leq j$, the longest suffix of $T[i..k]$ that is also a suffix of $T[i..j]$. Hence the list L of minimum suffix-centers of $T[i..j]$ can be constructed in $O(j-i)$ time.*

Proof. Note that there is an linear time algorithm to find, for all $1 \leq i \leq |P|$, the longest prefix of $P[i..n]$ that is also a prefix of P (see [5]). By reversing $T[i..j]$ and applying this algorithm, we can find, for all $i \leq s \leq j$, the longest suffix of $T[i..(s-1)]$ that is equal to some suffix of $T[i..j]$, and the lemma follows. \square

4.2 Checking the List L of Minimum Suffix-Centers

Recall that L is stored as a list of index pairs (s, e) . In this section, we show how to determine, for every $h \in \{j+1, j+2, \dots, 2j-i+1\}$, whether there is an

index pair $(s, e) \in L$ such that $T[s..e]$ equals $T[(j+1)..h]$. Note that checking the list L in brute force might take quadratic time. However, we can speedup the process drastically if we throw away the index pair $(s, e) \in L$ as soon as we find that it cannot be equal to $T[(j+1)..h]$ for any h . Here are the details.

After the first character $T[j+1]$ arrives, we scan through L . If there is a pair $(s, e) \in L$ where $s = e$ and $T[s] = T[j+1]$, we stop as we have found the first hanging square. Otherwise, we throw away all (s, e) with $T[s] \neq T[j+1]$ because $T[s..e]$ cannot be equal to $T[(j+1)..h]$ for any h ; their first characters are already different. Then, we repeat the process when the second character $T[j+2]$ is read. In general, when $T[j+x]$ ($j+x = h$) is read, we can be sure that for all remaining $(s, e) \in L$, the first $x-1$ characters of $T[s..e]$ are equal to $T[(j+1)..(j+x-1)]$. Thus if there is a $(s, e) \in L$ with $e = s+x-1$ and $T[e] = T[j+x]$, then we have found the first hanging square. Otherwise, we can throw away all (s, e) with $T[s+x-1] \neq T[j+x]$.

Note that when each character $T[h]$ is read, and when we scan L , we only check one character for each remaining $(s, e) \in L$. Thus, if we denote by P_x the set of remaining index pairs in L when we start to process $T[j+x]$, the running time of this checking process is

$$|P_1| + |P_2| + \cdots + |P_m| \quad (2)$$

where $T[j+m]$ ($j+m = h$), $m \leq j-i+1$ is the last character before we stop. It is easy to see, as shown in the lemma below, that $|P_x| \leq (j-i)/x$ for all $1 \leq x \leq m$.

Lemma 5. *For any $1 \leq x \leq m$, $|P_x| \leq (j-i)/x$. Thus, $|P_1| + |P_2| + \cdots + |P_m|$, the total time for checking L , is not more than $(j-i)(1 + 1/2 + \cdots + 1/m) = O((j-i) \log(j-i))$*

Proof. Note that for every pair $(s, e) \in P_x$, it has passed the first $x-1$ scans and thus the first x characters of $T[s..e]$ must be equal to $T[(j+1)..h]$. It follows that for any two $(s, e), (s', e') \in P_x$ (assume $s < s'$), the first x characters of $T[s..e]$ and $T[s'..e']$ are equal, i.e., $T[s..(s+x-1)] = T[s'..(s'+x-1)]$, (both are equal to $T[(j+1)..h]$). It is important to note that $T[s..(s+x-1)]$ and $T[s'..(s'+x-1)]$ are disjoint; otherwise, we have $s < s' < s+x-1 < s'+x-1$ and together with the fact that $T[s..(s+x-1)] = T[s'..(s'+x-1)]$, we conclude $T[s..(s'-1)] = T[s'..(2s'-s-1)]$ and $T[s..(2s'-s-1)]$ is a square; this is impossible because we would have stop earlier. Consequently, there can be at most $(j-i)/(x-1)$ index pairs in P_x , or equivalently, $|P_x| \leq (j-i)/(x-1)$. The lemma follows. \square

5 Future Work

We have designed an efficient online algorithm for detecting a square. We find that the bottleneck of our algorithm is the checking of the list of minimum suffix-centers. Currently, our checking is rather straightforward, and we believe

there are some better checking procedure that can help reduce the overall time complexity. Another interesting problem is to general our algorithm to detect cube, or even higher order of repetitions, in a string.

References

1. A. Apostolico and D. Breslauer. An optimal $O(\log \log n)$ -time parallel algorithm for detecting squares in a string. *SIAM Journal on Computing*, 25(6):1318–1331, 1996.
2. A. Apostolico and E.P. Preparata. Optimal off-line detection of repetitions in a string. *Theoretical Computer Science*, 22:297–315, 1983.
3. M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12(5):244–250, 1981.
4. M. Crochemore, W. Rytter, and R. Crochemore. *Text Algorithms*. Oxford University Press, 1994.
5. D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.
6. J.H.M. Lee, H.F. Leung, and H.W. Won. Extending genet for non-binary constraint satisfaction problems. In *Proceedings of the 7th International Conference on Tools with Artificial Intelligence*, pages 338–343, 1995.
7. J.H.M. Lee, H.F. Leung, and H.W. Won. Performance of a comprehensive and efficient constraint library based on local search. In *Proceedings of the 11th Australian Join Conference on Artificial Intelligence*, pages 13–17, 1998.
8. M.G. Main and R.J. Lorentz. An $O(n \log n)$ algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5:422–432, 1984.
9. M.G. Main and R.J. Lorentz. Linear time recognition of squarefree strings. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume F12 of *NATO ASI Series*, pages 271–278. Springer-Verlag, Berlin Heidelberg, 1985.
10. M.O. Rabin. Discovering repetitions in strings. *Combinatorial Algorithms on Words*, pages 279–288, 1985.
11. J.H.Y. Wong and H.F. Leung. Solving fuzzy constraint satisfaction problems with fuzzy genet. In *Proceedings of the 10th IEEE International Conference on Tools with Artificial Intelligence*, pages 180–191, 1998.

An Efficient Local Alignment Algorithm for Masked Sequences^{*} (Extended Abstract)

Jin Wook Kim and Kunsoo Park

School of Computer Science and Engineering,
Seoul National University, Seoul, 151-742, Korea
{jwkim,kpark}@theory.snu.ac.kr

Abstract. We consider the local alignment problem where sequences may have masked regions. The bases in masked regions are either unspecified or unknown, and they will be denoted by N. We present an efficient algorithm that finds an optimal local alignment by skipping such masked regions of sequences. Our algorithm works for both the affine gap penalty model and the linear gap penalty model. The time complexity of our algorithm is $O((n - T)(m - S) + vm + wn)$ time, where n and m are the lengths of given sequences a and b , T and S are the numbers of base N in a and b , and v and w are the numbers of masked regions of a and b , respectively.

1 Introduction

A local alignment algorithm finds substrings α and β of sequences a and b , respectively, where α is similar to β [7]. A local alignment algorithm is used in many applications such as DNA sequencing programs. When the lengths of a and b are n and m , respectively, Smith and Waterman [12] gave a well-known $O(n^2m)$ time local alignment algorithm for the affine gap penalty model and Gotoh [5] improved it to $O(nm)$ time. Crochemore et al. [2] gave a subquadratic time algorithm but it considers only a linear gap penalty model.

In DNA sequencing [1, 6, 8, 9, 13], some bases of fragments lose their information due to various reasons. RepeatMasker [11] screens DNA sequences for interspersed repeats and low complexity DNA sequences. Low-quality bases of DNA sequences are also screened by vector screening [1, 6, 9]. Some bases of fragments may not be determined by a base caller [3]. These *unspecified* or *unknown* bases will be denoted by N in this paper. Since masked regions lost their base information, alignments with masked regions are meaningless.

In this paper we present an efficient algorithm that finds an optimal local alignment by skipping such masked regions of sequences. In spite of skipping masked regions, the local alignment that our algorithm finds is the same as the one that the Smith-Waterman-Gotoh algorithm finds by computing all entries.

^{*} This work was supported by the MOST grant M1-0309-06-0003.

To skip masked regions, we first give a new recurrence for *insignificant entries* in a dynamic programming table, and then develop an efficient algorithm based on the new recurrence. Our algorithm runs in $O((n - T)(m - S) + vm + wn)$ time, where T and S are the numbers of base N in a and b , and v and w are the numbers of masked regions of a and b , respectively.

The remainder of the paper is organized as follows. In Section 2, we describe the local alignment algorithm due to Smith, Waterman, and Gotoh, and we extend it to include base N. In Section 3 we give a new recurrence for insignificant entries. In Section 4 we present a new local alignment algorithm for the affine gap penalty model that is based on the new recurrence. Section 5 gives the time complexity analysis of our algorithm. Section 6 briefly describes a simpler algorithm for the linear gap penalty model.

2 Preliminaries

We first give some definitions and notations that will be used in our algorithm. A string or a sequence is concatenations of zero or more characters from an alphabet Σ . A space is denoted by $\Delta \notin \Sigma$; we regard Δ as a character for convenience. The length of a string a is denoted by $|a|$. Let a_i denote i th character of a string a and $a_{i,j}$ denote a substring $a_i a_{i+1} \dots a_j$ of a . When a sequence α is a substring of a sequence a , we denote it by $\alpha \prec a$. Given two strings $a = a_1 a_2 \dots a_n$ and $b = b_1 b_2 \dots b_m$, an alignment of a and b is $a^* = a_1^* a_2^* \dots a_l^*$ and $b^* = b_1^* b_2^* \dots b_l^*$ constructed by inserting zero or more Δ s into a and b so that each a_i^* maps to b_i^* for $1 \leq i \leq l$. There are three kinds of mappings in a^* and b^* according to the characters of a_i^* and b_i^* .

- match : $a_i^* = b_i^* \neq \Delta$,
- mismatch : $(a_i^* \neq b_i^*)$ and $(a_i^*, b_i^* \neq \Delta)$,
- insertion or deletion (indel for short) : either a_i^* or b_i^* is Δ .

Note that we do not allow the case of $a_i^* = b_i^* = \Delta$.

2.1 Local Alignments

Given two sequences a and b , a local alignment of a and b is an alignment of two strings α and β such $\alpha \prec a$ and $\beta \prec b$, and an optimal local alignment of a and b is a local alignment of a and b that has the highest similarity among all local alignments of a and b . We denote the similarity of an optimal local alignment by $SL(a, b)$.

A well-known algorithm to find an optimal local alignment was given by Smith and Waterman [12], and Gotoh [5], which will be called the Smith-Waterman-Gotoh algorithm (SWG algorithm for short) [12, 14, 5]. Given two sequences a and b where $|a| = n$ and $|b| = m$, the SWG algorithm computes $SL(a, b)$ using a dynamic programming table (called the *H table*) of size $(n + 1)(m + 1)$. Let H_{ij} for $0 \leq i \leq n$ and $0 \leq j \leq m$ denote $SL(a_{1,i}, b_{1,j})$. Then, H_{ij} can be computed by the following recurrence:

$$H_{i,0} = H_{0,j} = 0 \quad \text{for } 0 \leq i \leq n, 0 \leq j \leq m$$

	A	C	G	T	N
A	1	-2	-2	-2	0
C	-2	1	-2	-2	0
G	-2	-2	1	-2	0
T	-2	-2	-2	1	0
N	0	0	0	0	0

Fig. 1. Scoring matrix for DNA sequence where $\delta = 2$ and $\sigma = 0$.

$$H_{ij} = \max \{H_{i-1,j-1} + s(a_i, b_j), C_{ij}, R_{ij}, 0\} \quad \text{for } 1 \leq i \leq n, 1 \leq j \leq m$$

where $C_{ij} = \max_{1 \leq k \leq i} \{H_{i-k,j} - g_k\}$ and $R_{ij} = \max_{1 \leq k \leq j} \{H_{i,j-k} - g_k\}$. $s(a_i, b_j)$ is a similarity between elements a_i and b_j such that

$$s(a_i, b_j) = \begin{cases} 1 & \text{if } a_i = b_j \\ -\delta & \text{if } a_i \neq b_j \end{cases}$$

and g_k is the gap penalty for an indel of $k \geq 1$ bases such that $g_k = \gamma + k\mu$ where δ , γ , and μ are non-negative constants. Among all H_{ij} for $0 \leq i \leq n$ and $0 \leq j \leq m$, the highest value is $SL(a, b)$, and the SWG algorithm takes $O(nm)$ time to compute it [5].

2.2 Scoring Matrix

In case of DNA sequences, there are four bases, A, C, G, and T. In addition, IUB ambiguity codes [10] shows several other bases which are for incompletely specified bases. In practice, however, bases N and X are only used. N means ‘unspecified’ and X means ‘unknown’. These bases are used when the original bases are screened by repeat masking or vector screening, or when the original bases are not determined by a base caller. In this paper we will use only N for an unspecified or unknown base.

We extend the definition of similarity $s(a_i, b_j)$. Because N might be any base, it is impossible to determine whether an alignment with N is a match or a mismatch. Hence, similarity $s(a_i, b_j)$ is newly defined such that

$$s(a_i, b_j) = \begin{cases} 1 & \text{if } a_i = b_j \neq N \\ -\delta & \text{if } a_i \neq b_j \text{ and } a_i, b_j \neq N \\ \sigma & \text{if either } a_i \text{ or } b_i \text{ is } N \end{cases}$$

where σ is a constant (which may not be 0) [4, 6]. Figure 1 shows an example of the scoring matrix for DNA sequence alignments where $\delta = 2$ and $\sigma = 0$. We call an entry H_{ij} where $a_i = N$ or $b_j = N$ an *insignificant entry*.

2.3 Gap Penalty Models [7]

We defined the gap penalty g_k as $g_k = \gamma + k\mu$ where γ and μ are non-negative constants. This is called the *affine gap penalty model*, where γ is the gap initiation penalty and μ is the gap extension penalty.

When there is no gap initiation penalty, i.e., $g_k = k\mu$, we call it the *linear gap penalty model*.

3 New Recurrence for an Insignificant Entry

In this section we present a new recurrence for an insignificant entry H_{ij} . We begin with some lemmas that are needed to get the new recurrence.

We define $C_{0j} = R_{i0} = -\infty$ for $1 \leq i \leq m$ and $1 \leq j \leq n$ and define $g_0 = 0$.

Let a and b be two DNA sequences where $|a| = n$ and $|b| = m$. Suppose that $a_{i-t+1,i}$ is a masked region of length t , i.e., $a_{i-t+1,i} = \text{NN} \dots \text{N}$.

Lemma 1. $H_{ij} \geq H_{i-y,j-x} + z\sigma - g_{x-z} - g_{y-z}$ for some constants x, y and z such that $0 \leq x \leq j$, $0 \leq y \leq t$, and $0 \leq z \leq \min\{x, y\}$.

The above lemma provides a relation between H_{ij} and its previous defined entries. If equality holds in Lemma 1, we say that H_{ij} comes from $H_{i-y,j-x}$. In other words, an optimal local alignment which has a mapping between a_i and b_j also has a mapping between a_{i-y} and b_{j-x} .

We will use ‘come from’ for some more cases. If $C_{ij} = H_{i-k,j} - g_k$ and $H_{i-k,j}$ comes from $H_{i-y,j-x}$, we say that C_{ij} comes from $H_{i-y,j-x}$. If $C_{i-y,j-x} = H_{i-y-k,j-x} - g_k$ and $H_{ij} = H_{i-y-k,j-x} + (y+k-t)\sigma - g_t - g_{x-y-k+t}$ for $t > k$, we say that H_{ij} comes from $C_{i-y,j-x}$. Similarly, we define that C_{ij} comes from $C_{i-y,j-x}$.

The following lemmas provide the range of entries which H_{ij} can come from.

Lemma 2. If H_{ij} comes from $H_{i-t,j-u}$ for $u \geq 0$, there exist at least one $H_{i-t,j-s}$ which H_{ij} comes from for some constant s such that $0 \leq s \leq t$.

Lemma 3. If H_{ij} comes from $C_{i-t,j-u}$ for $u \geq 0$, there exist at least one $C_{i-t,j-s}$ which H_{ij} comes from for some constant s such that $0 \leq s \leq t-1$.

The following lemmas provide the range of entries which C_{ij} can come from.

Lemma 4. If C_{ij} comes from $H_{i-t,j-u}$ for $u \geq 0$, there exist at least one $H_{i-t,j-s}$ which C_{ij} comes from for some constant s such that $0 \leq s \leq t-1$.

Lemma 5. If C_{ij} comes from $C_{i-t,j-u}$ for $u \geq 1$, there exist at least one $H_{i-t,j-s}$ which C_{ij} comes from for some constant s such that $0 \leq s \leq t-1$.

The following lemmas handle the case H_{ij} can come from the leftmost column. (For C_{ij} , results similar to Lemmas 6 and 7 hold.)

Lemma 6. If $t < j \leq m$ and H_{ij} comes from $H_{i-u,0}$ for $0 \leq u \leq t$, H_{ij} comes from $H_{i-t,j-s}$ for $0 \leq s \leq t$.

Lemma 7. If $0 < j \leq t$ and H_{ij} comes from $H_{i-u,0}$ for $0 \leq u \leq t$, H_{ij} comes from $H_{i-j,0}$.

By above lemmas, we derive the following theorem which provides a new recurrence for an insignificant entry H_{ij} .

Theorem 1. Let $a_{i-t+1,i}$ be a masked region of length t . Then H_{ij} can be computed by the following recurrence:

$$C_{ij} = \begin{cases} \max \left\{ \begin{array}{l} \max_{t-j+1 \leq k \leq t} \{H_{i-t,j-t+k} + (t-k)\sigma - \gamma - k\mu\}, \\ j\sigma - \gamma - \mu, C_{i-t,j} - t\mu \end{array} \right\} & \text{for } 1 \leq j < t \\ \max \left\{ \begin{array}{l} \max_{1 \leq k \leq t} \{H_{i-t,j-t+k} + (t-k)\sigma - \gamma - k\mu\}, \\ C_{i-t,j} - t\mu \end{array} \right\} & \text{for } t \leq j \leq m. \end{cases}$$

$$H_{ij} = \begin{cases} \max \left\{ \begin{array}{l} \max_{t-j+1 \leq k \leq t-1} \{C_{i-t,j-t+k} + (t-k)\sigma - k\mu\}, \\ j\sigma, C_{ij}, 0 \end{array} \right\} & \text{for } 1 \leq j \leq t \\ \max \left\{ \begin{array}{l} \max_{1 \leq k \leq t-1} \{C_{i-t,j-t+k} + (t-k)\sigma - k\mu\}, \\ H_{i-t,j-t} + t\sigma, C_{ij}, 0 \end{array} \right\} & \text{for } t < j \leq m. \end{cases}$$

Suppose that $b_{i-t+1,i}$ is a masked region of length t . The above lemmas and Theorem 1 hold in the symmetric case. In other words, they hold if we replace H_{ij} by H_{ji} and C_{ij} by R_{ji} .

4 New Algorithm for Affine Gap Penalty Model

A straightforward method to find an optimal local alignment of masked sequences a and b where $|a| = n$ and $|b| = m$ is to run the SWG algorithm in $O(nm)$ time. We want to find the local alignment without computing insignificant entries.

We present an algorithm for the affine gap penalty model. Our algorithm consists of the following four cases:

- Case 1: Neither a_i nor b_j is N. Compute all these entries by the SWG algorithm.
- Case 2: Only a_i is N. Compute the bottom row of a block of insignificant entries.
- Case 3: Only b_j is N. Compute the rightmost column of a block of insignificant entries.
- Case 4: Both a_i and b_j are N. Compute the bottom-right corner entry of a block of insignificant entries. In addition, compute a column to the left of the block and a row on top of the block. See Figure 2.

Figure 2 shows an example of four cases in the H table. We explain our algorithm for Case 2 and Case 4, since Case 3 is symmetric to Case 2.

4.1 Case 2

Let $a_{i-t+1,i}$ be a masked region of length t , i.e., $a_{i-t+1,i} = \text{NN} \dots \text{N}$. We will show how to compute the bottom row $H_{i,j-q+1..j}$ of a $t \times q$ block in $O(q)$ time. To do this, we first find the relation between two adjacent insignificant entries.

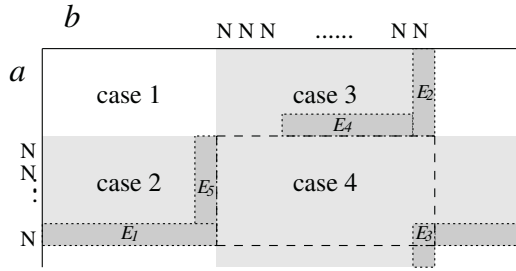


Fig. 2. An example of four cases in the H table. In Case 1, we compute all the entries. In Case 2, we compute only row E_1 . We compute only column E_2 in Case 3. In Case 4, we compute an entry E_3 and compute row E_4 and column E_5 .

Then, we define a new data structure, MQQUEUE, and give an algorithm of $O(q)$ time.

Computing the bottom row without computing other insignificant entries is possible by using the recurrences for an insignificant entry in Theorem 1. But it still takes $O(tq)$ time for a $t \times q$ block because it needs $O(t)$ computations for each C_{ij} and H_{ij} . However, we can reduce the time complexity to $O(q)$ using some properties of the recurrences.

Two adjacent entries of C are very similar. See Figure 3. For $t < j \leq m$, C_{ij} and $C_{i,j+1}$ are computed as

$$C_{ij} = \max\left\{\max_{1 \leq k \leq t}\{H_{i-t,j-t+k} + (t-k)\sigma - \gamma - k\mu\}, C_{i-t,j} - t\mu\right\},$$

$$C_{i,j+1} = \max\left\{\max_{1 \leq k \leq t}\{H_{i-t,j+1-t+k} + (t-k)\sigma - \gamma - k\mu\}, C_{i-t,j+1} - t\mu\right\}.$$

The differences between C_{ij} and $C_{i,j+1}$ are that (1) the element $H_{i-t,j-t+1} + (t-1)\sigma - \gamma - \mu$ is deleted, (2) $\sigma + \mu$ is added to all elements except the last of C_{ij} , (3) new element $H_{i-t,j+1} - \gamma - t\mu$ is inserted, (4) the element $C_{i-t,j} - t\mu$ is deleted, and (5) the element $C_{i-t,j+1} - t\mu$ is inserted. Thus $C_{i,j+1}$ can be rewritten as

$$C_{i,j+1} = \max\left\{\max_{2 \leq k \leq t+1}\{H_{i-t,j-t+k} + (t-k)\sigma - \gamma - k\mu\}, C_{i-t,j+1} - \sigma - (t+1)\mu\right\} + \sigma + \mu. \quad (1)$$

An insignificant entry $H_{i,j+1}$ can also be rewritten as follows.

$$H_{i,j+1} = \max\left\{\max_{2 \leq k \leq t}\{C_{i-t,j-t+k} + (t-k)\sigma - \gamma - k\mu\}, H_{i-t,j+1-t} + (t-1)\sigma - \mu, C_{i,j+1} - \sigma - \mu, -\sigma - \mu\right\} + \sigma + \mu.$$

Similarly, for $1 \leq j \leq t$, we can compute $C_{i,j+1}$ and $H_{i,j+1}$ using C_{ij} and H_{ij} , respectively.

We need a new data structure to find maximum values in $O(1)$ time. Because the element $H_{i-t,j-t+1} + (t-1)\sigma - \gamma - \mu$ is deleted in Recurrence (1),

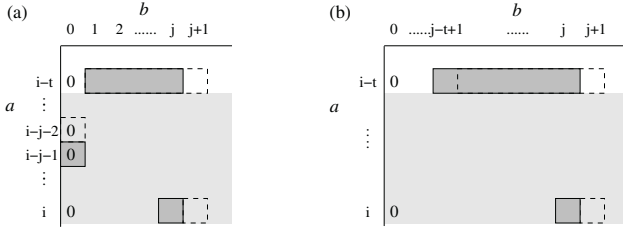


Fig. 3. (a) $1 \leq j \leq t$. To compute C_{ij} , we need $H_{i-j-1,0}$ and $H_{i-t,k}$ for $1 \leq k \leq j$ and to $C_{i,j+1}$, we need $H_{i-j-2,0} = 0$ and $H_{i-t,k}$ for $1 \leq k \leq j+1$. (b) $t < j \leq m$. To compute C_{ij} and $C_{i,j+1}$, we need $H_{i-t,j-t+k}$ and $H_{i-t,j+1-t+k}$, respectively, for $1 \leq k \leq t$.

$\max_{1 \leq k \leq t} \{H_{i-t,j-t+k} + (t-k)\sigma - \gamma - k\mu\}$ which is already computed in C_{ij} cannot be directly used in computing $C_{i,j+1}$. However, if we know the maximum value for the range of k such that $2 \leq k \leq t$, it can be used in $C_{i,j+1}$. We will find the maximum value using MQUEUE.

We define a specialized queue data structure, MQUEUE, that supports the following three operations:

- max()** : Return the maximum score among all elements in MQUEUE.
- insert(e)** : Add an element e to MQUEUE.
- delete()** : Remove the oldest element in MQUEUE.

The **max()** operation is added to the traditional queue.

To obtain $O(1)$ amortized time for each operation, MQUEUE maintains a list of maximum candidates. Maximum candidates are defined as follows: Given a set of ordered elements, the first maximum candidate is an element that has the maximum score in the list. If the i th maximum candidate is defined, then the $(i+1)$ st maximum candidate is an element that has the maximum score in the range from the next element of the i th maximum candidate to the last element of the list. Figure 4 (a) shows a list of maximum candidates.

The list of maximum candidates is maintained as a doubly linked list. A pointer *head* points to the maximum element of MQUEUE and *tail* points to the last one. The important property of the list is that the scores of the elements are in descending order.

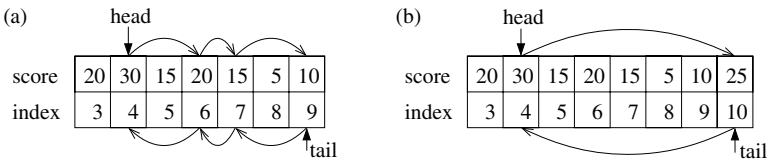


Fig. 4. An example of MQUEUE. (a) The list of maximum candidates is $30 \rightarrow 20 \rightarrow 15 \rightarrow 10$. (b) After **insert**(25), the list is modified to $30 \rightarrow 25$.

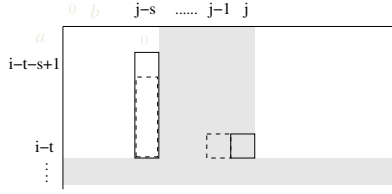


Fig. 5. Two MQUEUES for $H_{i-t,j}$ and $H_{i-t,j-1}$.

Operation `insert(e)` needs to maintain the list property. After adding an element e , the score of e is compared with the maximum candidates from *tail* until it meets the score that is larger than the score of e or the list becomes empty. When it meets a larger one, they link each other and *tail* points to e . When the list becomes empty, we make a new list which has one element e and *head* and *tail* point to e . Figure 4 (b) shows an example of `insert(e)`.

Operation `delete()` removes the oldest element from MQUEUE. If *head* points to the oldest element, then *head* is set to the next element in the list and we remove the oldest element. Operation `max()` returns the score of the element which is pointed to by *head*.

The algorithm for one block of insignificant entries with t rows consists of two **for** loops. The first **for** loop computes H_{ij} for $1 \leq j \leq t$ and the second **for** loop computes H_{ij} for $t+1 \leq j \leq m$. We use two MQUEUES, one for C_{ij} and the other for H_{ij} . In addition, we compute R_{ij} because it is used in Case 4.

4.2 Case 4

We will show how to compute the bottom-right corner entry H_{ij} of a $t \times s$ block and how to compute the row on top of the block and the column to the left of the block. Let $a_{i-t+1,i}$ and $b_{j-s+1,j}$ be masked regions, i.e., $a_{i-t+1,i} = \text{NN} \dots \text{N}$ and $b_{j-s+1,j} = \text{NN} \dots \text{N}$.

Lemma 8. *If H_{ij} comes from $C_{i-t,j-k}$ for $1 \leq k \leq \min\{s, t\} - 1$, H_{ij} comes from $C_{i-t,j}$ or $H_{i-t,j-u}$ for $0 \leq u \leq \min\{s, t\} - 2$.*

Lemma 9. *If $H_{i,j+r}$ comes from $C_{i-t,j+r-k}$ for $r+1 \leq k \leq \min\{\min\{s, t\} - 1 + r, t - 1\}$, $H_{i,j+r}$ comes from $C_{i-t,j}$ or $H_{i-t,j+r-u}$ for $r \leq u \leq \min\{\min\{s, t\} - 2 + r, t - 2\}$.*

Lemmas 8 and 9 show that in order to compute the bottom-right corner entry H_{ij} , we need $H_{i-t,j-k}$ for $0 \leq k \leq t$ and only $C_{i-t,j}$ (i.e., we don't need to compute $C_{i-t,j-k}$ for $1 \leq k \leq t$). C_{ij} also can be computed similarly to H_{ij} . Since $C_{i-t,j}$ is already computed in Case 3, we will show below how to compute $H_{i-t,j-k}$.

For $1 \leq k \leq \min\{t, s\} - 1$, we compute $H_{i-t,j-k}$ (row E_4 in Figure 2) using the MQUEUE that was constructed to compute $H_{i-t,j}$ in Case 3, i.e., we compute $H_{i-t,j-k}$ from right to left. See Figure 5. The difference between two MQUEUES

for $H_{i-t,j}$ and $H_{i-t,j-1}$ is that the oldest element, $R_{i-t-s+1,j-s}$, in MQUEUE for $H_{i-t,j}$ is deleted. Thus, after one **delete**, we have MQUEUE for $H_{i-t,j-1}$. Of course, to compute $H_{i-t,j-1}$, μ is added to **max**. Similarly, we can compute $H_{i-k,j-s}$ for $1 \leq k \leq \min\{t, s\} - 1$ (column E_5 in Figure 2).

The entries in row E_4 of Figure 2 will be used in Case 2 which is to the right of the current block. To compute the entry $H_{i,j+1}$ in Case 2, we need $H_{i-t,j+1-t..j+1}$ and $C_{i-t,j+1-t+1..j+1}$ among which $H_{i-t,j+1-t..j}$ and $C_{i-t,j+1-t+1..j}$ are the entries that are lain outside of the block of Case 2. We already have computed $H_{i-t,j+1-t..j}$ in Case 4 which is to the left block of the Case 2. For C , we need only $C_{i-t,j}$ by Lemma 9 and already have it. Similarly, the entries in column E_5 of Figure 2 will be used in Case 3 which is below the current block.

5 Analysis

Consider the time complexity for each of the four cases. In Case 1, the algorithm takes $O(pq)$ time for a $p \times q$ block because we use the SWG algorithm to compute H_{ij} . Since Case 3 is symmetric to Case 2, we consider Cases 2 and 4.

In Case 2, the algorithm takes $O(q)$ time for a $t \times q$ block. There are $q + 1$ **insert**, q **max** and $q - t$ **delete** operations. Because **max** and **delete** take constant time, q **max** and $q - t$ **delete** operations take $O(q)$ time. For **insert**, $q + 1$ **insert** operations add $q + 1$ elements. Moreover, each **insert** makes some comparisons to maintain the list of maximum candidates. If there are c comparisons in one **insert**, the number of elements in the list of maximum candidates is reduced by $c - 1$. Hence, each **insert** takes amortized $O(1)$ time and thus the total time for Case 2 is $O(q)$.

In Case 4, the algorithm takes $O(\min\{t, s\})$ time for a $t \times s$ block. Because it computes one row and one column with $\min\{t, s\}$ entries and the operations of MQUEUE take amortized $O(1)$ time, it takes $O(\min\{t, s\})$ time.

Now consider the worst case time complexity of our algorithm. The worst case is that the lengths of all masked regions of a and b are the same (say, t). Let v and w be the numbers of masked regions of a and b , respectively, and let T and S be the total numbers of N in a and b , respectively (i.e., $T = vt$, $S = wt$). Case 1 covers $(n - T)(m - S)$ entries and it takes $O((n - T)(m - S))$ time. Since Case 2 covers $vt \times (m - S)$ entries, Case 3 covers $wt \times (n - T)$ entries, and Case 4 covers $vt \times wt$ entries, they take $O(vm + wn)$ time. Therefore, the total time complexity is $O((n - T)(m - S) + vm + wn)$.

6 Algorithm for Linear Gap Penalty Model

Because the linear gap penalty model is a special case of the affine gap penalty model, it can be solved by the algorithm for the affine gap penalty model. From Lemmas 2, 6, and 7, however, we derive the following theorem which provides a simpler recurrence for an insignificant entry H_{ij} .

Theorem 2.

$$H_{ij} = \begin{cases} \max \left\{ j\sigma, \max_{t-j+1 \leq k \leq t} \{H_{i-t, j-t+k} + (t-k)\sigma - k\mu\}, 0 \right\} & \text{for } 1 \leq j \leq t \\ \max \left\{ \max_{0 \leq k \leq t} \{H_{i-t, j-t+k} + (t-k)\sigma - k\mu\}, 0 \right\} & \text{for } t < j \leq m. \end{cases}$$

Using the new recurrence of Theorem 2, we can give a local alignment algorithm for the linear gap penalty model that is similar to the algorithm for the affine gap penalty model, and the time complexity is also $O((n - T)(m - S) + vm + un)$.

References

1. A. Batzoglou, D. B. Jaffe, K. Stanley, J. Butler, et al. ARACHNE: A Whole-Genome Shotgun Assembler. *Genome Research*, 12:177–189, 2002
2. M. Crochemore, G.M. Landau, and M. Ziv-Ukelson. A Sub-quadratic Sequence Alignment Algorithm for Unrestricted Cost Matrices. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'02)*, 679–688, 2002
3. B. Ewing, L. Hillier, M. C. Wendl, and P. Green. Base-Calling of Automated Sequencer Traces Using Phred. I. Accuracy Assessment, *Genome Research*, 8:175–185, 1998
4. GCG Documentation, <http://www-igbmc.u-strasbg.fr/BioInfo/GCGdoc/Doc.html>
5. O. Gotoh. An Improved Algorithm for Matching Biological Sequences. *Journal of Molecular Biology*, 162:705–708, 1982
6. P. Green. PHRAP, <http://www.phrap.org>
7. D. Gusfield. *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, 1997
8. J. W. Kim, K. Roh, K. Park, et al. MLP: Mate-based Layout with PHRAP. In *Proc. 7th Annual International Conference on Research in Computational Molecular Biology - Currents in Computational Molecular Biology 2003*, 65–66, 2003
9. E. W. Myers, G. G. Sutton, A. L. Delcher, I. M. Dew, D. P. Fasulo, et al. A Whole-Genome Assembly of Drosophila. *Science*. 287:2196–2204, 2000
10. NC-UIB. Nomenclature for incompletely specified bases in nucleic acid sequences. Recommendations 1985. *The European Journal of Biochemistry*, 150:1–5, 1985
11. A. F. A. Smit and P. Green. <http://ftp.genome.washington.edu/RM/RepeatMasker.html>
12. T. F. Smith and M. S. Waterman. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147:195–197, 1981
13. J. Wang, G. K. Wong, P. Ni, et al. RePS: A Sequence Assembler That Masks Exact Repeats Identified from the Shotgun Data. *Genome Research*, 12:824–831, 2002
14. M. S. Waterman. *Introduction to Computational Biology*, Chapman & Hall, London, 1995

Computing Phylogenetic Roots with Bounded Degrees and Errors Is Hard^{*}

Tatsuie Tsukiji¹ and Zhi-Zhong Chen²

¹ Department of Information Science, Tokyo Denki University,
Hatoyama, Saitama 350-0394, Japan
tsukiji@j.dendai.ac.jp

² Department of Mathematical Sciences, Tokyo Denki University,
Hatoyama, Saitama 350-0394, Japan
chen@r.dendai.ac.jp

Abstract. The DEGREE- Δ CLOSEST PHYLOGENETIC k TH ROOT PROBLEM (Δ CPR k) is the problem of finding a (phylogenetic) tree T from a given graph $G = (V, E)$ such that (1) the degree of each internal node of T is at least 3 and at most Δ , (2) the external nodes (*i.e.* leaves) of T are exactly the elements of V , and (3) the number of disagreements, $|E \oplus \{\{u, v\} : u, v \text{ are leaves of } T \text{ and } d_T(u, v) \leq k\}|$ does not exceed a given number, where $d_T(u, v)$ denotes the distance between u and v in tree T . We show that this problem is NP-hard for all fixed constants $\Delta, k \geq 3$.

Our major technical contribution is the determination of all pylogenetic roots that approximate the almost largest cliques. Specifically, let $s_\Delta(k)$ be the size of a largest clique having a k th phylogenetic root with maximum degree Δ . We determine all the phylogenic k th roots with maximum degree Δ that approximate the $(s_\Delta(k) - 1)$ -clique within error 2, where we allow the internal nodes of phylogeny to have degree 2.

1 Introduction

A phylogeny is a tree where the leaves are labeled by species and each internal node represents a speciation event whereby an ancestral species gives rise to two or more child species. The internal nodes of a phylogeny have degrees (in the sense of unrooted trees, *i.e.* the number of incident edges) at least 3. Specifically, interspecies similarity is represented by a graph where the vertices are the species and the adjacency relation represents evidence of evolutionary similarity. A phylogeny is then reconstructed from the graph such that the leaves of the phylogeny are labeled by vertices of the graph (*i.e.* species) and for any two vertices of the graph, they are adjacent in the graph if and only if their corresponding leaves in the phylogeny are connected by a path of length at most k , where k is a predetermined proximity threshold. To be clear, vertices in the graph are called *vertices* while those in the phylogeny *nodes*. Recall that the length of the (unique) path connecting two nodes u and v in phylogeny T is the

^{*} Supported in part by the Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan, under Grant No. 14580390.

number of edges on the path, which is denoted by $d_T(u, v)$. This approach gives rise to the following algorithmic problem [5]:

PHYLOGENETIC k TH ROOT PROBLEM (PR k):

Given a graph $G = (V, E)$, find a phylogeny T with leaves labeled by the elements of V such that for each pair of vertices $u, v \in V$, $\{u, v\} \in E$ if and only if $d_T(u, v) \leq k$.

Such a phylogeny T (if exists) is called a *phylogenetic k th root*, or a *k th root phylogeny*, of graph G . Graph G is called the *k th phylogenetic power* of T . For convenience, we denote the *k th phylogenetic power* of any phylogeny T as $\mathcal{P}_k(T)$. That is, $\mathcal{P}_k(T)$ has the vertex set $L(T) = \{u : u \text{ are leaves of } T\}$ and the edge set $T^k = \{\{u, v\} \mid u \text{ and } v \text{ are leaves of } T \text{ and } d_T(u, v) \leq k\}$. Thus, PR k asks for a phylogeny T such that $G = \mathcal{P}_k(T)$.

The input graph in PR k is derived from some similarity data, which is usually inexact in practice and may have erroneous (spurious or missing) edges. Such errors may result in graphs that have no phylogenetic roots and hence we are interested in finding *approximate* phylogenetic roots for such graphs. For a graph $G = (V, E)$, each tree T whose leaves are exactly the elements of V is called an *approximate* phylogeny of G , and the *error* of T is $|T^k \oplus E| = |(E - T^k) \cup (T^k - E)|$. This motivated Chen *et. al.* to consider the following problem:

CLOSEST PHYLOGENETIC k TH ROOT PROBLEM (CPR k):

Given a graph $G = (V, E)$ and a nonnegative integer ℓ , decide if G has an approximate phylogeny T with at most ℓ errors.

An approximate phylogeny of G with the minimum number of errors is called a *closest k th root phylogeny* of graph G .

The hardness of PR k for large k seems to come from the unbounded degree of an internal node in the output phylogeny. On the other hand, in the practice of phylogeny reconstruction, most phylogenies considered are trees of degree 3 [7] because speciation events are usually bifurcating events in the evolutionary process. We call these restricted versions the DEGREE- Δ PR k and the DEGREE- Δ CPR k problems, and denote them for short as Δ PR k and Δ CPR k , respectively.

1.1 Previous Results on Phylogenetic Root Problems

PR k was first studied in [5] where linear-time algorithms for PR2 and PR3 were proposed. A linear-time algorithm for the special case of PR4 where the input graph is required to be connected, was also presented in [5]. At present, the complexity of PR k with $k \geq 5$ is still unknown.

Chen *et. al.* [2] presented a linear-time algorithm that determines, for any input *connected* graph G and constant $\Delta \geq 3$, if G has a *k th root phylogeny* with degree at most Δ , and if so, demonstrates one such phylogeny. On the other hand, Chen *et. al.* [2] showed that CPR k is NP-complete, for any $k \geq 2$. One of their open questions asks for the complexity of Δ CPR k .

Of special interest is CPR_2 . The problem CPR_2 is essentially identical to the correlation clustering problem which has drawn much attention recently [1]. The proof of the NP-hardness of CPR_2 given in [2] is also a valid proof of the NP-hardness of the correlation clustering problem. Blum *et. al.* [1] rediscovered the NP-hardness of the problem.

1.2 Our Contribution

In this paper, we will show that $\Delta\text{CPR}k$ is NP-complete, for any $k \geq 3$ and $\Delta \geq 3$. This answers an open question in [2].

In a course of the proof we first reduce HAMILTONIAN PATH, a famous NP-complete problem, to $\Delta\text{CPR}3$, and then $\Delta\text{CPR}3$ to $\Delta\text{CPR}k$. The former reduction is tedious but a routine work. On the other hand, the latter reduction seems to require new combinatorial investigation that is proper of $\Delta\text{CPR}k$.

A (Δ, k, h, ℓ) -core graph is a graph $G = (V, E)$ with the following properties:

- There is a tree T of maximum degree Δ whose phylogenetic k th power is G and such that T has a unique unsaturated (*i.e.* degree $< \Delta$) internal node α , the degree of α is $\Delta - 1$, $d_T(\alpha, u) = h$ holds for one leaf u and $d_T(\alpha, v) \geq h + 1$ hold for all leaves v other than u .
- For every approximate phylogeny T of G with maximum degree Δ and at most ℓ errors, there is at most one pair (α, u) such that α is an unsaturated internal node of T , u is a leaf of T , and $d_T(\alpha, u) \leq h$; moreover, if (α, u) exists then the degree of α in T is $\Delta - 1$.

Then, we establish the reduction from $\Delta\text{CPR}3$ to $\Delta\text{CPR}k$ by providing a family of $(\Delta, k, \lfloor k/2 \rfloor - 1, 2)$ -core graphs for every fixed $\Delta \geq 3$ and $k \geq 4$. Our construction of a $(\Delta, k, \lfloor k/2 \rfloor - 1, 2)$ -core graph is a pile of $(\Delta, k', 1, 2)$ -core graphs for $k' = 5, 7, \dots, k$ if $k \geq 5$ is odd, and $k' = 4, 6, \dots, k$ if $k \geq 4$ is even. So, a more basic problem is to prove that a certain graph is a $(\Delta, k, 1, 2)$ -core graph.

The maximum size of a clique having a (no-error) k th root phylogeny with maximum degree Δ is given by the following function,

$$s_\Delta(k) = \begin{cases} \Delta \cdot (\Delta - 1)^{\frac{k}{2} - 1}, & \text{if } k \text{ is even,} \\ 2 \cdot (\Delta - 1)^{\frac{k-1}{2}}, & \text{if } k \text{ is odd.} \end{cases}$$

We prove that the clique of size $s_\Delta(k) - 1$ is a $(\Delta, k, 1, 2)$ -core graph. Moreover, we determine the all k th root phylogenies with maximum degree Δ that approximate the clique within error 2, where we allow the internal nodes of phylogeny to have degree 2. For example, all phylogenetic roots of the $(s_3(5) - 1)$ -clique are D_5 in Figure 1, E_5 in Figure 2, and the tree obtained from D_5 by removing the leaf u .

2 Notations and Definitions

We employ standard terminologies in graph theory. In particular, for a graph G , $V(G)$ and $E(G)$ denote the sets of vertices and edges of G , respectively.

Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if there is a one-to-one correspondence φ between V and V' such that $\{u, v\} \in E$ if and only if $\{\varphi(u), \varphi(v)\} \in E'$, and we denote it as $G \cong_{\varphi} G'$. Let G be a graph. Let u and v be two vertices in G . The *distance* between u and v in G is the number of edges in a shortest path from u to v in G , and is denoted by $d_G(u, v)$. The *neighborhood* of v in G is the set of vertices adjacent to v in G , and is denoted by $N_G(v)$. The *degree* of v in G is $|N_G(v)|$, and is denoted by $d_G(v)$. Similarly, for a tree T , $V(T)$, $E(T)$, and $L(T)$ denote the sets of nodes, edges and leaves of T , respectively.

We also introduce some new terminologies of trees for convenience. For a tree T of maximum degree Δ , an internal node α of T is *unsaturated* if $d_T(\alpha) \leq \Delta - 1$. Tree T is *i -extensible* if $i = \sum_v (\Delta - \deg_T(v))$, where the summation is taken over all unsaturated internal nodes v of T . A tree T is *h -away* if for each unsaturated internal node x of T , the minimum distance from x to a leaf is at least h and further there is exactly one leaf v_x such that $d_T(x, v_x) = h$. For any set U of nodes of T , $T[U]$ denotes the minimum subtree containing U . Note that each leaf of $T[U]$ belongs to U . A phylogeny is a tree that contains no degree-2 nodes. As already mentioned, the k th phylogenetic power of any tree T is denoted as $\mathcal{P}_k(T) = (L(T), T^k)$, where T^k is the set of all edges $\{u, v\}$ with $\{u, v\} \subseteq L(T)$ and $d_T(u, v) \leq k$.

3 Construction of $(\Delta, k, \lfloor k/2 \rfloor - 1, 2)$ -Core Graphs

In this section we give a construction of $(3, k, \lfloor k/2 \rfloor - 1, 2)$ -core graphs for every odd $k \geq 5$. It is straightforward to generalize the arguments of this section to obtain $(\Delta, k, \lfloor k/2 \rfloor - 1, 2)$ -core graphs for every $\Delta \geq 3$ and $k \geq 4$.

Throughout this section, all trees and phylogenies are of maximum degree 3 or less. We abbreviate $s_3(k)$ as $s(k)$. The proofs of the most lemmas and corollaries are omitted due to lack of space.

Obviously, up to isomorphism, there is exactly one tree of maximum degree 3 whose k th phylogenetic power realizes an $s(k)$ -clique. We denote this tree by C_k . Similarly, up to isomorphism, there is exactly one tree of maximum degree 3 and having exactly one degree-2 node whose k th phylogenetic power realizes an $(s(k) - 1)$ -clique. We denote this tree by D_k . Figure 1 depicts D_4 , D_5 , and D_6 .

Lemma 1. *For every tree T (of maximum degree 3), if there are two leaves u and v with $d_T(u, v) = k$ and all leaves w of T have distance at most k from both u and v , then T is isomorphic to a subtree of C_k .*

Corollary 1. *For any tree T , if $d_T(u, v) \leq k$ for all leaves u and v , then T is isomorphic to a subtree of C_k .*

Fact 1 *For every tree T with $|L(T)| = s(k) - 1$ and $|T^k| \geq \binom{s(k)-1}{2} - 2$, we have $d_T(u, v) \leq k$ for all but at most two unordered pairs $\{u, v\}$ of leaves of T .*

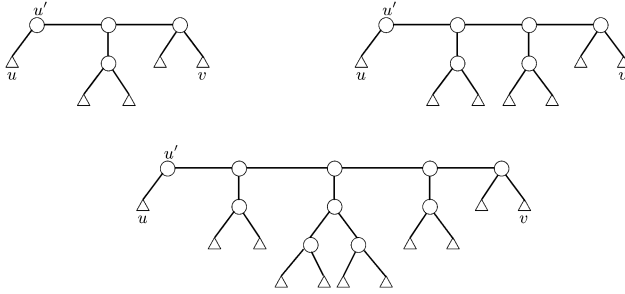


Fig. 1. D_4, D_5 and D_6 .

Lemma 2. Let $k \geq 4$. Let T be an arbitrary tree such that $|L(T)| = s(k) - 1$ and $|T^k| \geq \binom{s(k)-1}{2} - 2$. Then, there are leaves u and v of T with $d_T(u, v) = k$.

Lemma 3. Let $k \geq 6$. Let T be an arbitrary tree having $s(k) - 1$ leaves. Suppose that there are three leaves u, v, w of T such that $d_T(u, v) = k$ and further $\max(d_T(u, w), d_T(v, w)) \geq k + 1$. Then, $|T^k| \leq \binom{s(k)-1}{2} - 3$.

Lemma 4. Let $k \geq 6$. Let T be a tree having $s(k) - 1$ leaves such that $|T^k| \geq \binom{s(k)-1}{2} - 2$. Then T is 0-extensible or 1-extensible. Moreover, if T is 1-extensible then $T \cong D_k$.

Proof. By Lemma 2 we have two leaves u and v of T such that $d_T(u, v) = k$. Moreover, by Lemma 3, all leaves w of T partake distance at most k from both u and v . So, by Lemma 1, T is isomorphic to a subtree C'_k of C_k , where $|L(C'_k)| = |L(C_k)| - 1$. Since all internal nodes of C_k have degree 3, C'_k is obtained from C_k by removing one leaf, or two sibling leaves. In the former case, $T \cong D_k$, while in the latter case, T is 0-extensible.

For $k \in \{4, 5\}$, let E_k be the tree in Figure 2.

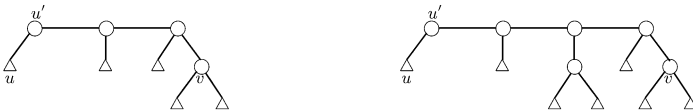


Fig. 2. E_4 and E_5 .

Lemma 5. Let $k \in \{4, 5\}$. Let T be a tree having $s(k) - 1$ leaves such that $|T^k| \geq \binom{s(k)-1}{2} - 2$. Then T is 0-extensible or 1-extensible. Moreover, if T is 1-extensible then $T \cong D_k$ or $T \cong E_k$.

Proof. By lemma 2, T has leaves u and v with $d_T(u, v) = k$. Let $S = \{w \in L(T) : d_T(u, w) \geq k + 1 \text{ or } d_T(v, w) \geq k + 1\}$. By assumption, $0 \leq |S| \leq 2$, and

Fact 1 holds. By Lemma 1, $T[L(T) - S]$ is isomorphic to a subtree C'_k of C_k . Let φ be the isomorphism. Root T at v and C_k at $\varphi(v)$. Let u'' be the grandparent of $\varphi(u)$ in C_k . The proof proceeds in three cases.

Case 1: $|S| = 0$, i.e. $S = \emptyset$. In this case, $T \cong C'_k$. So, either $T \cong D_k$ or T is 0-extensible.

Case 2: $|S| = 1$. Let $S = \{w\}$. Without loss of generality, $d_T(u, w) \geq d_T(v, w)$. Let w' be the lowest ancestor of w with $d_T(w') = 3$.

Subcase 2.1: w is not a descendant of $\varphi^{-1}(u'')$. Root E_k at the non-leaf node adjacent to v . We show that $T \cong E_k$, where leaf u of E_k corresponds to leaf w of T , the uncle of u in E_k corresponds to leaf v of T , and a child leaf of v in E_k corresponds to u (see Figure 2). First note that for all leaves x of C_k that are descendants of u'' , if $x \in C'_k$ then $d_T(x, w) \geq k + 1$. By Fact 1, at most two of them can belong to $L(C'_k)$. Moreover, at least one leaf descendant of $\varphi(w')$ in C_k does not belong to $L(C'_k)$. Now, since $|L(T) - \{w\}| = |L(C'_k)| = s(k) - 2$, we have that the sibling and uncle of u belong to $L(C'_k)$, and $d_T(u, w) = k + 1$. Consequently, $T \cong E_k$.

Subcase 2.2: w is a descendant of $\varphi^{-1}(u'')$. Since $d_T(u, w) \geq d_T(v, w)$, we have $k = 4$ and $d_T(v, w) = d_T(u, w) \geq k + 1 = 5$. By Fact 1, the leaves of C_4 that are of distance 2 from either $\varphi(u)$ or $\varphi(v)$ do not belong to C'_4 . Moreover, so does at least one descendant of $\varphi(w')$. Therefore, $s(4) - 2 = |L(T) - \{w\}| = |L(C'_4)| \leq s(4) - 3$, a contradiction.

Case 3: $|S| = 2$. Let $S = \{w, y\}$. In this case, we show that $T \cong E_k$, where leaf u of E_k corresponds to that of T , the leaf of E_k whose distance from v is 2 corresponds to leaf v of T and the two leaves of E_k adjacent to v correspond to w, y (see Figure 2). We may assume $d_T(u, w) \geq d_T(v, w)$. Then, w is not a descendant of $\varphi^{-1}(u'')$, otherwise we could get a contradiction as in Subcase 2.2. Similarly, y is not a descendant of $\varphi^{-1}(u'')$. Let w' be the lowest ancestor of w in T that is a node of $T[L(T) - S]$. We already have two nodes in S that have distance greater than k from u or v in T . So, by Fact 1, none of the sibling nor cousins of $\varphi(u)$ belong to $L(C'_k)$. In addition, so does at least one leaf descendant of $\varphi(w')$. Thus, $s(k) - 3 = |L(T) - S| = |L(C'_k)|$. Consequently, the uncle of $\varphi(u)$ belongs to $L(C'_k)$, w and y are the only descendants of w' that belong to $L(C_k) - L(C'_k)$, and $d_T(u, w) = d_T(u, y) = k + 1$. So, $T \cong E_k$.

Corollary 2. *For every $k \geq 4$, an $(s(k) - 1)$ -clique is a $(3, k, 1, 2)$ -core graph.*

Now we are ready to construct a $(3, k, \lfloor k/2 \rfloor - 1, 2)$ -core graph for every odd $k \geq 5$. We recursively construct trees $R_{k, \lfloor k/2 \rfloor - 1}$, $k = 5, 7, 9, \dots$, and define a family of $(3, k, \lfloor k/2 \rfloor - 1, 2)$ -core graphs as the k th phylogenetic power of the trees $R_{k, \lfloor k/2 \rfloor - 1}$ (see Figure 3 for $R_{7, 2}$):

- Let $h_k = \lfloor k/2 \rfloor - 1$. For each $1 \leq i \leq h_k$, let $g(i) = \prod_{j=1}^i (s(2j+3) - 1)$. Let $g(0) = 1$.
- \tilde{R}_{k, h_k} is a leveled tree of depth h_k such that at depth i ($0 \leq i \leq h_k$) are placed $g(i)$ nodes and each node v at depth $i < h_k$ is connected to some $s(2i+5) - 1$ nodes at depth $i + 1$.

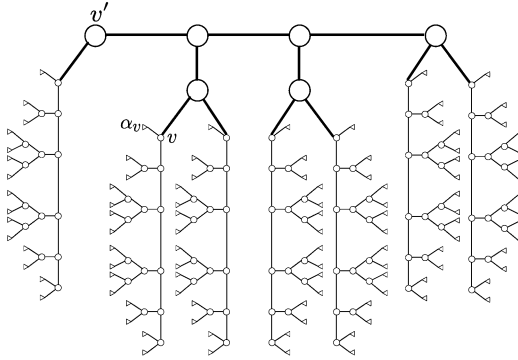


Fig. 3. $R_{7,2}$.

- R_{k,h_k} is an expansion of \tilde{R}_{k,h_k} such that each node v of \tilde{R}_{k,h_k} at depth i ($0 \leq i \leq h_k - 1$) is expanded to a copy $D(v)$ of D_{2i+5} in R_{k,h_k} , where v is identified with the degree-2 node of $D(v)$ and the child nodes of v in \tilde{R}_{k,h_k} are identified with the leaves of $D(v)$ in an arbitrary one-to-one manner.

By construction, R_{k,h_k} is 1-extensible and h_k -away, and has a unique degree-2 node, namely, the unique degree-2 node of D_5 .

Lemma 6. *Let $k \geq 4$. Let T be a 0-extensible tree, having $s(k) - 1$ leaves, and $|T^k| \geq \binom{s(k)-1}{2} - 2$. Let F be the tree obtained by connecting a new leaf to a leaf w of T . Then, $|F^k| \leq \binom{s(k)-1}{2} - 3$.*

Lemma 7. *Let $k \geq 5$ be odd. Let T be a tree such that $L(T) = L(R_k)$ and $|T^k \oplus R_k^k| \leq 2$. Then, T is 0-extensible or 1-extensible. Moreover, if T is 1-extensible then it is h_k -away.*

Proof. By induction on $k \geq 5$. The case $k = 5$ has been done in Lemma 5, because $R_5 = D_5$. So we fix $k \geq 7$. Let $\tilde{R} = \tilde{R}_k$, $R = R_k$ and $h = h_k$, for simplicity of notations. Let Γ_i be the set of nodes of \tilde{R} at depth i . Let $L_v = L(D(v))$ for every non-leaf node v of \tilde{R} . So, $L(R) = \Gamma_h = \cup_{v \in \Gamma_{h-1}} L_v$. Consider an arbitrary tree T such that $L(T) = L(R)$ and $|T^k \oplus R^k| \leq 2$. Then, for every $v \in \Gamma_{h-1}$, $\mathcal{P}_k(R[L_v])$ is an $(s(k) - 1)$ -clique and $|T^k[L_v] \oplus R^k[L_v]| \leq 2$, so by Lemma 4, $T[L_v]$ is 1-extensible and 1-away. Let $\varphi(v)$ be the degree-2 node of $T[L_v]$. Let α_v (respectively, β_v) be the unique leaf of $D(v)$ (respectively, $T[L_v]$) such that $d_R(v, \alpha_v) = 1$ (respectively, $d_T(\varphi(v), \beta_v) = 1$). We may assume T is not 0-extensible.

We claim that $\alpha_v = \beta_v$ for all $v \in \Gamma_{h-1}$. For a contradiction, we assume $\alpha_v \neq \beta_v$ for some $v \in \Gamma_{h-1}$. Let $v' \in \Gamma_{h-2}$ be the parent node of v in \tilde{R} . By construction, $\mathcal{P}_{k-2}(R[L_{v'}])$ is an $(s(k-2)-1)$ -clique, so $\mathcal{P}_k(R[\{\alpha_u : u \in L_{v'}\}])$ is an $(s(k-2)-1)$ -clique. Since $|T^k \oplus R^k| \leq 2$, $|T^k[\{\alpha_u : u \in L_{v'}\}]| \geq \binom{s(k-2)-1}{2} - 2$. In addition, $d_T(\varphi(u), \alpha_u) \geq 1$ for all $u \in L_{v'}$, so if we let $T_0 = T[\{\varphi(u) : u \in L_{v'}\}]$, then $|T_0^{k-2}| \geq \binom{s(k-2)-1}{2} - 2$. Lemma 4 (or Lemma 5 for $k = 7$) therefore

determines the topology of T_0 . Particularly, T_0 is not 0-extensible; otherwise the subtree $T[\cup_{v \in L_{v'}} L_v]$ would become 0-extensible since every $T[L_v]$ with $v \in L_{v'}$ is 1-extensible. Let F be the tree obtained by connecting a new leaf to $\varphi(v)$. By Lemma 6, $|F^k| \leq \binom{s(k)-1}{2} - 3$, while we have also $|F^{k-2}| \geq \binom{s(k-2)-1}{2} - 2$ since $\alpha_v \neq \beta_v$ and $|T^k[\{\alpha_u : u \in L_{v'}\}]| \geq \binom{s(k-2)-1}{2} - 2$, a contradiction.

Now, $\alpha_v = \beta_v$ for all $v \in \Gamma_{h-1}$. Define a tree T' with $L(T') = \Gamma_{h-1}$ from T as follows: For every $v' \in \Gamma_{h-1}$, contract $T[L_v]$ to $\varphi(v)$ and identify $\varphi(v)$ with v . Because $d_T(\varphi(v), \alpha_v) = 1$ for all $v \in \Gamma_{h-1}$, $d_{T'}(u, v) = d_T(\alpha_u, \alpha_v) - 2$ for all $u, v \in \Gamma_{h-1}$. In addition, $d_R(u, v) = d_{R[\Gamma_{h-1}]}(\alpha_u, \alpha_v) - 2$, hence $|(T')^{k-2} \oplus R^{k-2}[\Gamma_{h-1}]| \leq 2$. The induction hypothesis therefore shows that T' is 1-extensible and h_{k-2} -away. Moreover, all $T[L_v]$, $v \in \Gamma_{h-1}$, are 1-extensible and 1-away, so T is 1-extensible and h_k -away.

Theorem 1. *For every odd $k \geq 5$, the graph $\mathcal{P}_k(R_{k, \lfloor k/2 \rfloor - 1})$ is a $(3, k, \lfloor k/2 \rfloor - 1, 2)$ -core graph.*

These constructions, lemmas and theorems can be generalized to every fixed $k \geq 4$ and $\Delta \geq 3$. A phylogenic k th root of the $s_\Delta(k)$ -clique can be constructed in the same way as D_i and we denote it as $D_{\Delta, i}$. We can construct a phylogeny R_{Δ, k, h_k} of degree Δ recursively in the same way as R_{k, h_k} but replacing s and D_i therein with s_Δ and $D_{\Delta, i}$, respectively; further, if k is even then the function $g(i)$ therein should be replaced by $\prod_{j=1}^i (s_\Delta(2j+2) - 1)$. Lemma 7 and Theorem 1 can be generalized as follows:

Theorem 2. *Let $k \geq 4$ and $\Delta \geq 3$. Let T be a tree of maximum degree Δ such that $L(T) = L(R_{\Delta, k, h_k})$ and $|T^k \oplus R_{\Delta, k, h_k}^k| \leq 2$. Then T is 0-extensible or 1-extensible. Moreover, if T is 1-extensible then it is h_k -away.*

Theorem 3. *For every $k \geq 4$, $\mathcal{P}_k(R_{\Delta, k, \lfloor k/2 \rfloor - 1})$ is a $(\Delta, k, \lfloor k/2 \rfloor - 1, 2)$ -core graph.*

4 The NP-Hardness of $\Delta\text{CPR}k$

This section proves that $3\text{CPR}k$ is NP-hard for each odd $k \geq 3$. It is straightforward to generalize the arguments of this section to prove that $\Delta\text{CPR}k$ is NP-hard for every $\Delta \geq 3$ and $k \geq 3$.

Throughout this section, all trees and phylogenies are of maximum degree 3 or less. Proofs of most lemmas and corollaries are omitted due to lack of space.

We begin with the NP-hardness proof of $3\text{CPR}3$ because the NP-hardness proofs of $3\text{CPR}k$ for larger odd k are reductions from it. We reduce the following version of HAMILTONIAN PATH PROBLEM (HP) to $3\text{CPR}3$, whose NP-hardness proofs can be found in [3] and [6, Section 9.3].

HAMILTONIAN PATH PROBLEM (HP): Given a graph $G = (V, E)$ such that

- all vertices are of degree 3 or less,
- two specific vertices are of degree 1 and each of them is adjacent to a vertex of degree 2, and
- there is no cycle of length less than 5,

find a Hamiltonian path of G , i.e., find a linear ordering of the vertices of G such that each pair of consecutive vertices are adjacent in G .

For every graph $G = (V, E)$ and every approximate phylogeny T of G , we define a function $f_{G,T}$ mapping each $v \in V$ to a real number as follows:

$$f_{G,T}(v) = \frac{1}{2} |\{u : \{u, v\} \in E \text{ and } d_T(u, v) > 3\}| \\ + |\{\{u, w\} : u \neq w, \{u, v\} \in E, \{v, w\} \in E, \{u, w\} \notin E \text{ and } d_T(u, w) \leq 3\}|.$$

Lemma 8. $\sum_{v \in V} f_{G,T}(v) \leq |T^3 \oplus E|$.

Lemma 9. Let v be a vertex of G having three pairwise nonadjacent neighbors u_1, u_2 and u_3 . Then, $f_{G,T}(v) = \frac{1}{2}$ or $f_{G,T}(v) \geq 1$. Moreover, if $f_{G,T}(v) = \frac{1}{2}$, then $d_T(u_i, v) > 3$ for one $u_i \in \{u_1, u_2, u_3\}$ and $d_T(u_j, v) = 3$ for the other two $u_j \in \{u_1, u_2, u_3\} - \{u_i\}$.

Lemma 10. 3CPR3 is NP-complete.

Proof. Let $G = (V, E)$ be an arbitrary instance of HP. Let $|V| = n + 2$. Let $W \subseteq V$ be the set of vertices of degree 3 in G . Then $|W|$ is even and we denote it as 2ℓ . Let v_0 and v_{n+1} be the two degree-1 vertices, and let v_1 and v_n be the unique neighbors of v_0 and v_{n+1} in G , respectively. Note that $d_G(v_1) = d_G(v_n) = 2$.

Let $G' = (V', E')$ be the graph obtained from G as follows.

1. Add two new vertices \tilde{v}_0 and \tilde{v}_{n+1} .
2. Add four new edges $\{v_0, \tilde{v}_0\}$, $\{\tilde{v}_0, v_1\}$, $\{v_{n+1}, \tilde{v}_{n+1}\}$, and $\{\tilde{v}_{n+1}, v_n\}$.

It suffices to show that the input graph G has a Hamiltonian path if and only if G' has an approximate phylogeny T with error $\leq \ell$.

Suppose that G has a Hamiltonian path $v_0, v_1, v_2, \dots, v_n, v_{n+1}$, an ordering of the vertices where each pair of consecutive vertices are adjacent in G . An approximate phylogeny T of G' is then constructed as follows: T has

1. $n + 2$ internal nodes v'_0, \dots, v'_{n+1} ,
2. $n + 4$ edges connecting the leaves to the internal nodes, namely, $\{\tilde{v}_0, v'_0\}$, $\{\tilde{v}_{n+1}, v'_{n+1}\}$ and $\{v_i, v'_i\}$ for all $i \in \{0, \dots, n + 1\}$, and
3. $n + 1$ edges $\{v'_i, v'_{i+1}\}$ for all $i \in \{0, \dots, n\}$ between the internal nodes.

By the construction, $|T^3 \oplus E'| = |\{\{v_i, v_j\} \in E : |i - j| > 1\}| = \ell$.

Conversely, suppose that T is an approximate phylogeny of G' that satisfies $|T^3 \oplus E'| \leq \ell$. By Lemma 8, $|T^3 \oplus E'| \geq \sum_{v \in V'} f_{G',T}(v) \geq \sum_{v \in W} f_{G',T}(v)$. On the other hand, by Lemma 9, $f_{G',T}(v) \geq \frac{1}{2}$ for all $v \in W$. So, $\ell \geq |E' \oplus T^3| \geq |W|/2 = \ell$, where all inequalities must be equality. This shows that,

- (i) $f_{G',T}(v) = \frac{1}{2}$ for all $v \in W$, and
- (ii) $f_{G',T}(v) = 0$ for all $v \in V' - W$.

For each $v_i \in V$, let v'_i be the internal node in T adjacent to v_i . We claim that v'_0, \dots, v'_{n+1} are different internal nodes of T . For a contradiction, assume

$v'_i = v'_j$ for some $v_i \neq v_j$. A contradiction is derived in each of the following cases.

Case 1: $v_i \in W$. Let u_1, u_2 and u_3 be the three neighbors in G . If $v_j \in \{u_1, u_2, u_3\}$, then $f_{G',T}(v_i) \geq 1$ by Lemma 9. Otherwise, either $u'_i = u'_j$, or both $d_T(u_i, v_i) > 3$ and $d_T(u_j, v_i) > 3$ for some $u_i \neq u_j$. In any case, $f_{G',T}(v_i) \geq 1$, a contradiction against (i).

Case 2: $v_i \in V - W - \{v_0, v_{n+1}\}$. Let u_1 and u_2 be the neighbors of v_i . If $v_j \in \{u_1, u_2\}$, say $u_2 = v_j$, and $d_T(u_1, u_2) \geq 4$, then $f_{G',T}(v_i) \geq \frac{1}{2}$. If $u_2 = v_j$ and $d_T(u_1, u_2) \leq 3$, then $f_{G',T}(v_i) \geq 1$. If $v_j \notin \{u_1, u_2\}$, then either $u'_1 = u'_2$ or at least one of $d_T(u_1, v_i)$ or $d_T(u_2, v_i)$ is larger than 3; In either case, $f_{G',T}(v_i) \geq \frac{1}{2}$. Therefore, in any case, we get a contradiction against (ii).

Case 3: $v_i = v_0$ and $v_j = v_{n+1}$. In this case, $d_T(v_0, v_1) \geq 4$ or $d_T(v_n, v_{n+1}) \geq 4$ holds. If $d_T(v_0, v_1) \geq 4$, then $f_{G',T}(v_0) \geq \frac{1}{2}$. Similarly, if $d_T(v_0, v_{n+1}) \geq 4$, then $f_{G',T}(v_{n+1}) \geq \frac{1}{2}$. In any case, we get a contradiction against (ii).

Now, $|\{v'_0, \dots, v'_{n+1}\}| = n + 2$. So, $d_T(v_i, v_j) \geq 3$ for all $v_i \neq v_j$. By Lemma 9 and (i), for every $v_i \in W$, v'_i is adjacent to just two of the three nodes u'_1, u'_2, u'_3 in T where u_1, u_2, u_3 are the neighbors of v_i in G . Moreover, by (ii), for every $v_i \in V - W - \{v_0, v_{n+1}\}$, v'_i is adjacent to both of u'_1 and u'_2 where u_1 and u_2 are the neighbors of v_i in G . Consequently, the subtree $T[\{v'_0, \dots, v'_{n+1}\}]$ becomes a Hamiltonian path starting from v'_0 and ending at v'_{n+1} , which gives rise to a Hamiltonian path from v_0 to v_{n+1} in G .

Theorem 4. *For every odd $k \geq 3$, 3CPR k is NP-complete.*

Proof. We have shown it for $k = 3$. Fix an arbitrary odd $k \geq 5$. By Theorem 1, we have a $(3, k, h_k, 2)$ -core graph $\mathcal{P}_k(R_{k,h_k})$ and its k th phylogenetic root R_{k,h_k} , where $h_k = \lfloor k/2 \rfloor - 1$. Recall that R_{k,h_k} has a unique degree-2 node α and there is a unique leaf u in R_{k,h_k} such that $d_{R_{k,h_k}}(\alpha, u) = h_k$. Moreover, for all leaves $v \neq u$ of R_{k,h_k} , $d_T(\alpha, v) > h_k$. We refer to this leaf u as the *special* leaf of R_{k,h_k} and the corresponding vertex of $\mathcal{P}_k(R_{k,h_k})$ as the *special* vertex of $\mathcal{P}_k(R_{k,h_k})$.

Let $G = (V, E)$ be an instance of HP. We inherit the notions used in the proof of Lemma 10 (e.g. $n = |V| - 2$ and 2ℓ is the number of degree-3 vertices in G .) Let $G' = (V', E')$ be the graph constructed from G as in Lemma 10. Further, we construct a graph $G'' = (V'', E'')$ from G' as follows:

1. Replace every vertex v in V' with a copy $G(v)$ of $\mathcal{P}_k(R_k)$ and identify v with the special vertex of $G(v)$, referring to it as the special vertex of $G(v)$.
2. Let $E'' \supseteq E'$, i.e. connect a pair of special vertices by an edge of G'' if and only if the corresponding pair are adjacent in G' .

Therefore, $V' \subseteq V''$ is the set of special vertices of G'' .

It suffices to show that the input graph G has a Hamiltonian path if and only if G'' has an approximate phylogeny with error $\leq \ell$.

Suppose that G has a Hamiltonian path $v_0, v_1, v_2, \dots, v_n, v_{n+1}$, an ordering of the vertices of G . Let T be the approximate phylogeny of G' constructed in Lemma 10 from this Hamiltonian path. A required approximate phylogeny T'' of G'' is then reconstructed as follows:

1. Replace each leaf $v \in V'$ of T with a copy $T(v)$ of tree R_{k,h_k} and connect the degree-2 node of $T(v)$ to the node adjacent to v in the original T .
2. Label the leaves of $T(v)$ by the vertices of $G(v)$ in an arbitrary one-to-one manner such that $\mathcal{P}_k(T(v)) = G(v)$ and v is the special leaf of $T(v)$.

Conversely, let T be an arbitrary approximate phylogeny of G'' with error $\leq \ell$. Our goal is to show the existence of a Hamiltonian path of G .

Let $L_v = V(G(v))$. We define a function f_k mapping each $v \in V'$ to a real number as follows:

$$\begin{aligned} f_k(v) = & \frac{1}{2} |\{u \in N_{G'}(v) : d_T(u, v) > k\}| \\ & + |\{\{u, w\} : u \neq w, v \in N_{G'}(u) \cap N_{G'}(w), \{u, w\} \notin E' \text{ and } d_T(u, w) \leq k\}| \\ & + \sum_{u \in N_{G'}(v) \cup \{v\}} \frac{|T^k[L_u] \oplus E(G(u))|}{d_{G'}(u) + 1}. \end{aligned}$$

Lemmas 8 and 9 still hold after replacing the function $f_{G,T}$ therein by the function f_k here. The proofs remain the same.

We claim that $f_k(v) \geq \frac{1}{2}$ for every degree-3 vertex v of G . For a contradiction, assume $f_k(v) < \frac{1}{2}$. Let u_1, u_2, u_3 be the neighbors of v in G . For every u_i , $|T^k[L_{u_i}] \oplus E(G(u_i))| \leq 2$; otherwise we would have $f_k(v) \geq \frac{3}{4}$. So by Theorem 1, $T[L_{u_i}]$ is 1-extensible and h_k -away. Similarly, $T[L_v]$ is 1-extensible and h_k -away. By definitions, these four 1-extensible subtrees $T[L_{u_1}], T[L_{u_2}], T[L_{u_3}]$ and $T[L_v]$ are mutually disjoint. Let α_i be the degree-2 node of $T[L_{u_i}]$ and α'_i be its neighbor outside $T[L_{u_i}]$. Similarly, let β be the degree-2 node of $T[L_v]$ and β' be its neighbor outside $T[L_v]$. Then, $|\{\alpha_1, \alpha_2, \alpha_3, \beta\}| = 4$, and by arguments in the proof of Lemma 9, $|\{\alpha'_1, \alpha'_2, \alpha'_3, \beta'\}| = 4$. We have $d_T(u_i, \alpha_i) = h_k$ for every u_i ; otherwise $d_T(v, u_i) \geq k+1$ hence we would have $f_k(v) \geq \frac{1}{2}$. Similarly, $d_T(v, \beta) = h_k$. Therefore, $\mathcal{P}_3(T)[\{\alpha_1, \alpha_2, \alpha_3, \beta\}] \cong \mathcal{P}_k(T)[\{u_1, u_2, u_3, v\}]$. Consequently, by the proof of Lemma 9, $f_k(v) \geq \frac{1}{2}$, a contradiction.

Now, $f_k(v) \geq \frac{1}{2}$ for every degree-3 vertex v of G . Together with the upper bound, we have

- (i) $f_k(v) = \frac{1}{2}$ for all vertices v of degree-3 in G and,
- (ii) $f_k(u) = 0$ holds for all other vertices u in V' .

Hence $T[L_v]$ is 1-extensible and h_k -away for every $v \in V'$. Let α_v be the unique degree-2 node of the subtree $T[L_v]$. We have shown that $d_T(v, \alpha_v) = h_k$ for all degree-3 vertices v of G ; further, by (i) and (ii), $d_T(v, \alpha_v) = h_k$ for all $v \in V'$. Thus, $\mathcal{P}_3(T[\{\alpha_v : v \in V'\}]) \cong \mathcal{P}_k(T[V'])$. Let T'' be the tree obtained from T by contracting every $T[L_v]$ to α_v and identifying α_v with v , for every $v \in V'$. Then, $|T''^3 \oplus E'| \leq \ell$; we know that G has a Hamiltonian path, as in the proof of Lemma 10.

It is straightforward to generalize Theorem 4 to every $\Delta \geq 3$ and $k \geq 4$, obtaining the following theorem.

Theorem 5. *For every $\Delta \geq 3$ and every $k \geq 3$, $\Delta\text{CPR}k$ is NP-complete.*

5 Summary and an Open Question

We have proved that $\Delta\text{CPR}k$ is NP-complete for every $\Delta \geq 3$ and $k \geq 3$. A more fundamental problem is the TREE k TH ROOT PROBLEM ($\text{TR}k$), where the nodes (not only the leaves) of T correspond to the vertices of G . Kearney and Corneil proved that $\text{CTR}k$ is NP-complete when $k \geq 3$ [4]. We conjecture that $\Delta\text{CTR}k$ is NP-complete for every fixed $\Delta \geq 3$ and $k \geq 2$.

References

1. N. BANSAL, A. BLUM, AND S. CHAWLA, *Correlation Clustering*, in Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS 2002), pages 238–250, 2002.
2. Z.-Z. CHEN, T. JIANG, AND G.-H. LIN, *Computing phylogenetic roots with bounded degrees and errors*, SIAM Journal on Computing, 32(4):864–879, 2003. A preliminary version appeared in Proceedings of WADS2001.
3. M. R. GAREY, D. S. JOHNSON AND R. E. TARJAN, *The Planar Hamiltonian Circuit Problem is NP-Complete*, SIAM Journal on Computing, 5(4):704–714, 1976.
4. P. E. KEARNEY AND D. G. CORNEIL, *Tree powers*, Journal of Algorithms, 29:111–131, 1998.
5. G.-H. LIN, P. E. KEARNEY, AND T. JIANG, *Phylogenetic k -root and Steiner k -root*, in The 11th Annual International Symposium on Algorithms and Computation (ISAAC 2000), volume 1969 of Lecture Notes in Computer Science, pages 539–551, Springer, 2000.
6. C. H. PAPADIMITRIOU, *Computational Complexity*, Addison-Wesley, 1994.
7. D. L. SWOFFORD, G. J. OLSEN, P. J. WADDELL, AND D. M. HILLIS, *Phylogenetic inference*, In D. M. Hillis, C. Moritz, and B. K. Mable, editors, Molecular Systematics (2nd Edition), pages 407–514, Sinauer Associates, Sunderland, Massachusetts, 1996.

Inferring a Level-1 Phylogenetic Network from a Dense Set of Rooted Triplets

Jesper Jansson and Wing-Kin Sung

School of Computing, National University of Singapore, 3 Science Drive 2,
Singapore 117543

{jansson,ksung}@comp.nus.edu.sg

Abstract. Given a set \mathcal{T} of rooted triplets with leaf set L , we consider the problem of determining whether there exists a phylogenetic network consistent with \mathcal{T} , and if so, constructing one. If no restrictions are placed on the hybrid nodes in the solution, the problem is trivially solved in polynomial time by a simple sorting network-based construction. For the more interesting (and biologically more motivated) case where the solution is required to be a level-1 phylogenetic network, we present an algorithm solving the problem in $O(n^6)$ time when \mathcal{T} is dense (i.e., contains at least one rooted triplet for each cardinality three subset of L), where $n = |L|$. Note that the size of the input is $\Theta(n^3)$ if \mathcal{T} is dense. We also give an $O(n^5)$ -time algorithm for finding the set of *all* phylogenetic networks having a single hybrid node attached to exactly one leaf (and having no other hybrid nodes) that are consistent with a given dense set of rooted triplets.

1 Introduction

A phylogenetic network is a generalization of a phylogenetic tree in which internal nodes are allowed to have more than one parent. Phylogenetic networks are used to represent evolutionary relationships that cannot be adequately described in a single tree structure due to evolutionary events such as recombination, horizontal gene transfer, or hybrid speciation which suggest convergence between objects [8, 9, 17, 18, 20]. In fact, these types of events occur frequently in the evolutionary history of certain groups of organisms [17], but not much is known about the combinatorics related to phylogenetic networks [8]. Hence, to develop conceptual tools and efficient methods for computing with phylogenetic networks is an important issue.

Some methods for constructing and for comparing phylogenetic networks have been proposed recently [4, 8, 17, 18, 20]. In this paper, we consider the problem of constructing a phylogenetic network from a set of rooted triplets (see below for a formal problem definition). In particular, we assume that the input forms a *dense* set, meaning that the input contains at least one rooted triplet for each cardinality three subset of the objects being studied, and that the underlying phylogenetic network is a *level-1* network, meaning that each biconnected component in the undirected version of the network contains at most one node

which has two parents in the directed version of the network. The biological significance of level-1 phylogenetic networks, there referred to as *galled-trees*, is discussed in [8]. The rationale for assuming the input to consist of rooted triplets is that although computationally expensive methods for constructing reliable phylogenetic trees such as maximum likelihood are infeasible for large sets of objects, they can be applied to infer highly accurate trees for smaller, overlapping subsets of the objects (see, e.g., [3]). One may thus apply maximum likelihood to each cardinality three subset L' of the objects and then select the most likely rooted triplet for L' to get a dense input set¹. Moreover, in some applications, the data obtained experimentally may already have the form of rooted triplets; for example, Sibley-Ahlquist-style DNA-DNA hybridization experiments (see [15]) can yield rooted triplets directly.

1.1 Definitions

A *rooted triplet* is a binary, rooted, unordered tree with three distinctly labeled leaves. The unique rooted triplet on leaf set $\{x, y, z\}$ in which the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of x and z (or equivalently, where the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of y and z) is denoted by $(\{x, y\}, z)$. A set \mathcal{T} of rooted triplets is called *dense* if for each $\{x, y, z\} \subseteq L$, where L is the set of all leaves occurring in \mathcal{T} , at least one of the three possible rooted triplets $(\{x, y\}, z)$, $(\{x, z\}, y)$, and $(\{y, z\}, x)$ belongs to \mathcal{T} .

A *phylogenetic network* is a connected, rooted, simple, directed acyclic graph in which: (1) each node has outdegree at most 2; (2) each node has indegree 1 or 2, except the root node which has indegree 0; (3) no node has both indegree 1 and outdegree 1; and (4) all nodes with outdegree 0 are labeled by elements from a finite set L in such a way that no two nodes are assigned the same label. From here on, nodes of outdegree 0 are referred to as *leaves* and identified with their corresponding elements in L . We refer to nodes with indegree 2 as *hybrid nodes*. For any phylogenetic network N , let $\mathcal{U}(N)$ be the undirected graph obtained from N by replacing each directed edge by an undirected edge. N is a *level- f phylogenetic network* if every biconnected component in $\mathcal{U}(N)$ contains at most f nodes that are hybrid nodes in N . Note that if $f = 0$ then N is a tree.

We denote the set of leaves in a rooted triplet t or a phylogenetic network N by $\Lambda(t)$ or $\Lambda(N)$, respectively. A rooted triplet t is *consistent with* the phylogenetic network N if t is an induced subgraph of N . See Fig. 1 for an example. A set \mathcal{T} of rooted triplets is *consistent with* N if every $t_i \in \mathcal{T}$ is consistent with N .

The problem we consider in this paper is: Given a set $\mathcal{T} = \{t_1, \dots, t_k\}$ of rooted triplets, construct a level-1 phylogenetic network N with $\Lambda(N) =$

¹ A similar approach is used in the quartet method paradigm [14, 16] for reconstructing unrooted phylogenetic trees: first infer the unrooted topology of each cardinality four subset of the leaf set to obtain a complete set of quartets (unrooted, distinctly leaf-labeled trees each having four leaves and no nodes of degree two), then combine the quartets into an unrooted tree.

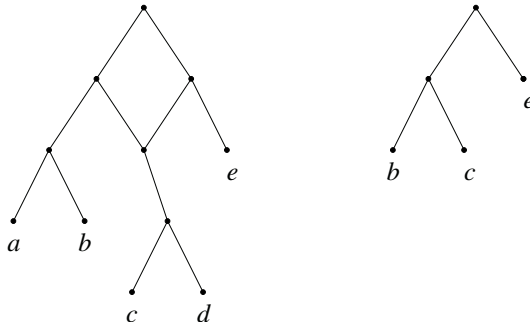


Fig. 1. Let N be the phylogenetic network on the left. The rooted triplet $(\{b, c\}, e)$ shown on the right is consistent with N . Note that $(\{c, e\}, b)$ is also consistent with N .

$\bigcup_{t_i \in \mathcal{T}} A(t_i)$ such that \mathcal{T} is consistent with N , if such a network exists; otherwise, output *null*. Throughout this paper, L represents the leaf set $\bigcup_{t_i \in \mathcal{T}} A(t_i)$ in the problem definition above, and we write $n = |L|$ and $k = |\mathcal{T}|$. Note that $\binom{n}{3} \leq k \leq 3 \cdot \binom{n}{3}$, i.e., $k = \Theta(n^3)$ if the input is dense.

Finally, for any set \mathcal{T} of rooted triplets and $L' \subseteq L$, we define $\mathcal{T} \upharpoonright L'$ as the subset of \mathcal{T} consisting of all rooted triplets $(\{x, y\}, z)$ with $\{x, y, z\} \subseteq L'$.

1.2 Related Work

Aho, Sagiv, Szymanski, and Ullman [1] presented an $O(kn)$ -time algorithm for determining whether a given set of k rooted triplets on n leaves is consistent with some rooted, distinctly leaf-labeled tree (i.e., a level-0 phylogenetic network), and if so, returning such a tree. Several years later, Henzinger, King, and Warnow [10] showed how to implement the algorithm of Aho *et al.* to run in $\min\{O(kn^{0.5}), O(k + n^2 \log n)\}$ time. Gąsieniec, Jansson, Lingas, and Östlin [6] considered a version of the problem where the leaves in the output tree are required to comply with a left-to-right leaf ordering given as part of the input. Related optimization problems where the objective is to construct a rooted tree consistent with the maximum number of rooted triplets in the input or to find a maximum cardinality subset L' of L such that $\mathcal{T} \upharpoonright L'$ is consistent with some tree have been studied in [2, 6, 7, 12] and [13], respectively.

The analog of the problem considered by Aho *et al.* for *unrooted* trees is NP-hard, even if all of the input trees are quartets [19]. Fortunately, certain useful optimization problems involving quartets can be approximated efficiently [14, 16]. For a survey on quartet-based methods for inferring unrooted phylogenetic trees and related computational complexity results, see [16].

Nakhleh, Warnow, and Linder [17] gave an algorithm for reconstructing a level-1 phylogenetic network from two distinctly leaf-labeled, binary, rooted, unordered trees with identical leaf sets. It runs in time which is polynomial in the number of leaves and the number of hybrid nodes in the underlying phylogenetic

network. They also considered the case where the two input trees may contain errors but where only one hybrid node is allowed.

We remark that the deterministic algorithm for dynamic graph connectivity employed in the algorithm of Henzinger *et al.* [10] mentioned above can in fact be replaced with a more recent one due to Holm, de Lichtenberg, and Thorup [11] to yield the following improvement.

Lemma 1. [13] *The algorithm of Aho et al. can be implemented to run in $\min\{O(k \log^2 n), O(k + n^2 \log n)\}$ time.*

1.3 Our Results and Organization of the Paper

We observe that if no restriction is placed on the level of the phylogenetic network, then the problem can be trivially solved using a sorting network-based construction in Section 2. Next, in Section 3, we present an $O(n^5)$ -time algorithm called *OneHybridLeaf* for inferring all phylogenetic networks with one hybrid node to which exactly one leaf is attached that are consistent with a given dense set of rooted triplets. This algorithm is subsequently used in Section 4, where we give a more general algorithm called *LevelOne* for constructing a level-1 phylogenetic network consistent with \mathcal{T} (if one exists) in $O(n^6)$ time when \mathcal{T} is dense.

2 Constructing an Unrestricted Phylogenetic Network

Given a set \mathcal{T} of rooted triplets, we can always construct a level- f phylogenetic network N where f is unrestricted such that N is consistent with \mathcal{T} . Moreover, the construction can be carried out in time which is polynomial in the size of \mathcal{T} as follows. Let P be any sorting network (see, e.g., [5]) for n elements with a polynomial number p of comparator stages. Build a directed acyclic graph Q from P with $(p+1) \cdot n$ nodes $\{Q_{i,j} \mid 1 \leq i \leq p+1, 1 \leq j \leq n\}$ such that there is a directed edge $(Q_{i,j}, Q_{i+1,j})$ for every $1 \leq i \leq p$ and $1 \leq j \leq n$, and two directed edges $(Q_{i,j}, Q_{i+1,k})$ and $(Q_{i,k}, Q_{i+1,j})$ for every comparator (j, k) at stage i in P for $1 \leq i \leq p$. Then, for $1 \leq j \leq n-1$, add the directed edge $(Q_{1,j}, Q_{1,j+1})$. See Fig. 2. Finally, distinctly label the nodes $\{Q_{p+1,j} \mid 1 \leq j \leq n\}$ by L , and for each node in Q having indegree 1 and outdegree 1 (if any), contract its outgoing edge to obtain N . It is easy to show that for any $\{x, y, z\} \subseteq L$, all three of $(\{x, y\}, z)$, $(\{x, z\}, y)$, and $(\{y, z\}, x)$ are consistent with N .

3 Constructing All Phylogenetic Networks Having One Hybrid Node with One Attached Leaf

This section presents an algorithm called *OneHybridLeaf* for inferring the set of all phylogenetic networks having a single hybrid node attached to exactly one leaf (and having no other hybrid nodes) which are consistent with a given set \mathcal{T} of rooted triplets. This algorithm is later used as a subroutine by the main algorithm in Section 4. *OneHybridLeaf* assumes that its given set \mathcal{T} of rooted triplets is dense. We first note the following.

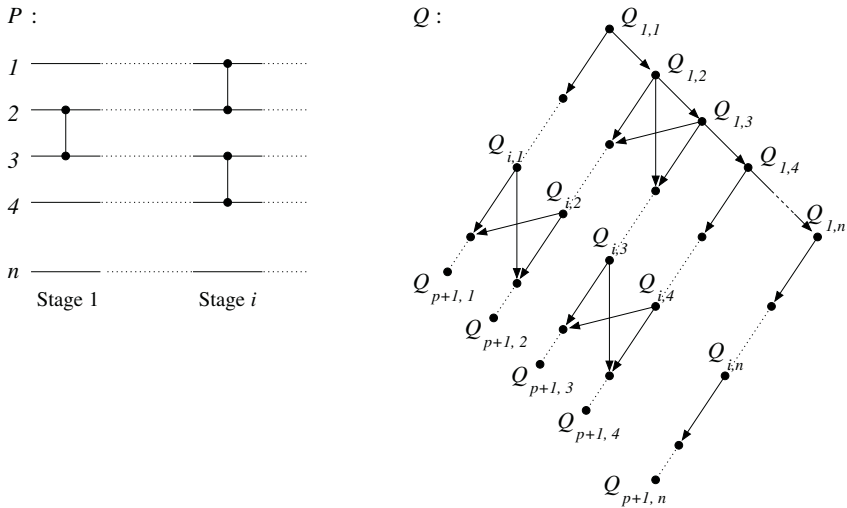


Fig. 2. The sorting network P on the left yields a directed acyclic graph Q .

Lemma 2. Let N be any phylogenetic network consistent with a set \mathcal{T} of rooted triplets with leaf set L such that N has a hybrid node h to which exactly one leaf c is attached and N has no other hybrid nodes. If h and c and all their incident edges are deleted and then, for every node with outdegree 1 and indegree less than 2, its outgoing edge is contracted, then the resulting graph is a binary tree consistent with $\mathcal{T} \mid (L \setminus \{c\})$.

Lemma 3. Let \mathcal{T} be a dense set of rooted triplets and let L be the leaf set of \mathcal{T} . There is at most one rooted, unordered tree distinctly leaf-labeled by L which is consistent with \mathcal{T} . Furthermore, if such a tree R exists then it must be binary.

Proof. Suppose there exist two unordered, distinctly leaf-labeled trees R and R' consistent with \mathcal{T} such that $R \neq R'$. Then, for some $x, y, z \in L$, $(\{x, y\}, z)$ is consistent with R while $(\{x, z\}, y)$ is consistent with R' . Since \mathcal{T} is dense, at least one of $(\{x, y\}, z)$, $(\{x, z\}, y)$, and $(\{y, z\}, x)$ belongs to \mathcal{T} . This yields a contradiction in all cases because R cannot be consistent with $(\{x, z\}, y)$ or $(\{y, z\}, x)$ and R' cannot be consistent with $(\{x, y\}, z)$ or $(\{y, z\}, x)$ since R and R' are trees.

Next, suppose R is not binary. Then R has a node u with degree greater than two. Let x, y , and z be leaves from three different subtrees rooted at children of u . \mathcal{T} is dense, so at least one of $(\{x, y\}, z)$, $(\{x, z\}, y)$, and $(\{y, z\}, x)$ belongs to \mathcal{T} . But none of these three rooted triplets is consistent with R . Contradiction. \square

Our algorithm *OneHybridLeaf* is shown in Fig. 3. It tests every $c \in L$ as the leaf attached to the hybrid node. For each such candidate c , it first calls a procedure *BuildTree* to obtain a binary tree R which is consistent with all rooted triplets in \mathcal{T} that do not involve the leaf c , if such a tree exists. (\mathcal{T} is dense, so the set $\mathcal{T} \mid (L \setminus \{c\})$ is also dense. Thus, Lemma 3 ensures that if R exists then it

Algorithm *OneHybridLeaf*

Input: A dense set \mathcal{T} of rooted triplets with leaf set L .

Output: The set of all phylogenetic networks which are consistent with \mathcal{T} having exactly one hybrid node to which there is exactly one leaf attached.

```

1  Set  $\mathcal{N} = \emptyset$ .
2  for each  $c \in L$  do
2.1  Let  $R = \text{BuildTree}(\mathcal{T} \mid (L \setminus \{c\}))$ .
2.2  if  $R \neq \text{null}$  then
2.3    for every pair of nodes  $u$  and  $v$  in  $R$ , where  $u \neq v$  and  $v$  is not the
        root of  $R$  do
2.3.1      Form a phylogenetic network  $N$  from  $R$  as follows. Create three
            new internal nodes  $p$ ,  $q$ , and  $h$ , insert the four edges  $(p, u)$ ,  $(p, h)$ ,
             $(q, v)$ , and  $(q, h)$ , and attach a leaf child labeled by  $c$  to  $h$ . Replace
            the edge  $(v_0, v)$  leading to  $v$  by the edge  $(v_0, q)$ . If  $u$  is not the root
            then replace  $u$ 's incoming edge  $(u_0, u)$  by the edge  $(u_0, p)$ .
2.3.2      If  $N$  is consistent with all rooted triplets in  $\mathcal{T}$  involving the leaf  $c$ 
            then let  $\mathcal{N} = \mathcal{N} \cup \{N\}$ .
        endfor
    endfor
3  return  $\mathcal{N}$ .
End OneHybridLeaf

```

Fig. 3. Constructing phylogenetic networks with one hybrid node.

is uniquely determined and binary.) Then, it tries all possible ways to obtain a phylogenetic network from R by inserting a hybrid node h attached to the leaf c , and keeps all resulting networks which are also consistent with the rest of \mathcal{T} . By Lemma 2, all valid networks will be found by *OneHybridLeaf*.

To implement *BuildTree*, we use the fast version of the algorithm of Aho *et al.* referred to in Lemma 1. If L contains at least four elements then $\text{BuildTree}(\mathcal{T} \mid (L \setminus \{c\}))$ is the algorithm of Aho *et al.* applied to $\mathcal{T} \mid (L \setminus \{c\})$ (we may assume it returns *null* if it fails). For the case $|L| = 3$, the set $\mathcal{T} \mid (L \setminus \{c\})$ is empty and we simply let $\text{BuildTree}(\mathcal{T} \mid (L \setminus \{c\}))$ return a tree with the two leaves in $L \setminus \{c\}$.

Lemma 4. *The time complexity of Algorithm OneHybridLeaf is $O(n^5)$.*

Proof. Step 2 iterates Steps 2.1–2.3 n times. In each iteration, Step 2.1 takes $O(k + n^2 \log n)$ time by Lemma 1. The inner **for**-loop (Step 2.3) considers $O(n^2)$ pairs of nodes of R ; for each such node pair, Step 2.3.1 takes $O(1)$ time and Step 2.3.2 takes $O(n^2)$ time. In total, Step 2.3 uses $O(n^2 \cdot (1 + n^2)) = O(n^4)$ time, so Step 2 takes $O(n \cdot (k + n^2 \log n + n^4))$ time. Furthermore, $k = |\mathcal{T}| = O(n^3)$. Thus, the total running time of Algorithm *OneHybridLeaf* is $O(n^5)$. \square

4 Constructing a Level-1 Phylogenetic Network

Here, we present an algorithm called *LevelOne* for inferring a level-1 phylogenetic network (if one exists) consistent with a given dense set \mathcal{T} of rooted triplets. The

basic idea of our algorithm is to partition the leaf set of \mathcal{T} into disjoint subsets which we call *SN*-sets, run *LevelOne* recursively to construct a level-1 network for each *SN*-set, and then apply Algorithm *OneHybridLeaf* from Section 3 to combine the computed networks for the *SN*-sets into one level-1 network.

We first introduce the concept of an *SN*-set. Let L be the leaf set of \mathcal{T} . For any $X \subseteq L$, define the set $SN(X)$ recursively as $SN(X \cup \{c\})$ if there exists some $c \in L \setminus X$ and $x_1, x_2 \in X$ such that $(\{x_1, c\}, x_2) \in \mathcal{T}$, and as X otherwise. Below, we study some properties of the *SN*-sets.

Lemma 5. *$SN(\{x, y\})$ for any $x, y \in L$ is computable in $O(n^3)$ time.*

Proof. If $x = y$ then $SN(\{x, y\}) = \{x\}$ can be obtained in $O(1)$ time. If $x \neq y$ then $SN(\{x, y\})$ can be computed by calling Algorithm *ComputeSN*(x, y) shown in Fig. 4. Initially, the algorithm sets $X = \{x\}$ and $Z = \{y\}$. Then, while Z is nonempty, it selects any $z \in Z$, augments Z with all leaves c not already in $X \cup Z$ such that $(\{a, c\}, z)$ or $(\{z, c\}, a) \in \mathcal{T}$ for some $a \in X$, and finally removes z from Z and inserts z into X . To analyze the time complexity of Algorithm *ComputeSN*, observe that one leaf is transferred from Z to X in each iteration of the **while**-loop and that a leaf which has been moved to X can never be moved back to Z , so Steps 2.1–2.3 are iterated at most $n - 1$ times. Inside the **while**-loop, the algorithm scans $O(n^2)$ rooted triplets at most once to augment Z . The total running time of *ComputeSN* is therefore $O(n^3)$. \square

Algorithm *ComputeSN*

Input: A dense set \mathcal{T} of rooted triplets with leaf set L . Two leaves $x, y \in L$.

Output: $SN(\{x, y\})$.

```

1  Set  $X = \{x\}$  and  $Z = \{y\}$ .
2  while  $Z \neq \emptyset$  do
2.1    Let  $z$  be any element in  $Z$ .
2.2    for every  $a \in X$  do
           If there exists some  $c \in L \setminus (X \cup Z)$  such that  $(\{a, c\}, z) \in \mathcal{T}$  or
            $(\{z, c\}, a) \in \mathcal{T}$  then  $Z = Z \cup \{c\}$ .
        endfor
2.3    Set  $X = X \cup \{z\}$  and  $Z = Z \setminus \{z\}$ .
    endwhile
3  return  $X$ .
End    ComputeSN
```

Fig. 4. Computing $SN(\{x, y\})$.

Lemma 6. *If \mathcal{T} is dense then for any $A, B \subseteq L$, $SN(A) \cap SN(B)$ equals \emptyset , $SN(A)$, or $SN(B)$.*

Proof. Suppose on the contrary that $z_1, z_2 \in SN(A)$, $z_2, z_3 \in SN(B)$, $z_3 \notin SN(A)$, and $z_1 \notin SN(B)$. Consider the rooted triplet on the three leaves z_1, z_2 , and z_3 . Since \mathcal{T} is dense, at least one of the following three cases must occur:

- Case 1: $(\{z_2, z_3\}, z_1) \in \mathcal{T}$. Then, by definition, $z_3 \in SN(A)$.
- Case 2: $(\{z_1, z_3\}, z_2) \in \mathcal{T}$. Then, by definition, $z_3 \in SN(A)$.
- Case 3: $(\{z_1, z_2\}, z_3) \in \mathcal{T}$. Then, by definition, $z_1 \in SN(B)$.

In each of the three cases, we have a contradiction. Thus, the lemma follows. \square

In particular, Lemma 6 holds for all subsets of L of cardinality one or two.

For any $x_1, x_2 \in L$ (possibly with $x_1 = x_2$), $SN(\{x_1, x_2\})$ is called *trivial* if $SN(\{x_1, x_2\}) = L$, and *maximal* if it is nontrivial and not a proper subset of any nontrivial $SN(\{y_1, y_2\})$, where $y_1, y_2 \in L$. Let \mathcal{SN} be the set of all maximal SN -sets of the form $SN(\{x_1, x_2\})$, where possibly $x_1 = x_2$. Since \mathcal{T} is dense, \mathcal{SN} forms a partition of the set L by Lemma 6. Furthermore, \mathcal{SN} is uniquely determined. Write $\mathcal{SN} = \{SN_1, SN_2, \dots, SN_q\}$ and introduce q new symbols $\alpha_1, \alpha_2, \dots, \alpha_q$. (Observe that $q \geq 2$ if $|L| \geq 2$.) We define a function f as follows. For every $x \in L$, let $f(x) = \alpha_i$ if $x \in SN_i$. Let \mathcal{T}' be the set $\{(\{f(x), f(y)\}, f(z)) : (\{x, y\}, z) \in \mathcal{T} \text{ and } f(x), f(y), f(z) \text{ all differ}\}$.

Lemma 7. *Suppose \mathcal{T} is consistent with a level-1 phylogenetic network. If $q = 2$ then the tree distinctly leaf-labeled by α_1 and α_2 is consistent with \mathcal{T}' . If $q \geq 3$ then there exists a phylogenetic network having a single hybrid node attached to exactly one leaf (and having no other hybrid nodes) that is consistent with \mathcal{T}' .*

We also have:

Lemma 8. *Suppose \mathcal{T}' is consistent with a level-1 phylogenetic network N' with leaf set $\{\alpha_1, \dots, \alpha_q\}$. Let N be a level-1 network obtained from N' by replacing each α_i by a level-1 network N_i with leaf set SN_i consistent with $\mathcal{T} \upharpoonright SN_i$. Then N is consistent with \mathcal{T} .*

Proof. Let t be any rooted triplet in \mathcal{T} and write $t = (\{x, y\}, z)$. If $x \in SN_i$, $y \in SN_j$, and $z \in SN_k$, where i, j, k all differ, then t is consistent with N (otherwise, $t' = (\{f(x), f(y)\}, f(z)) = (\{\alpha_i, \alpha_j\}, \alpha_k)$ cannot be consistent with N' which is a contradiction since $t' \in \mathcal{T}'$). If $x, y \in SN_i$ and $z \in SN_j$ with $i \neq j$ then t is consistent with N by the construction of N . The case $x, z \in SN_i$ and $y \in SN_j$ (or symmetrically, $y, z \in SN_i$ and $x \in SN_j$) with $i \neq j$ is not possible since $x, z \in SN_i$ implies $y \in SN_i$. If x, y, z belong to the same SN_i then t is consistent with N_i and therefore with N . In all cases, t is consistent with N . \square

Our main algorithm *LevelOne* is listed in Fig. 5. Its correctness follows from Lemmas 7 and 8.

Theorem 1. *When \mathcal{T} is dense, we can determine if there exists a level-1 phylogenetic network consistent with \mathcal{T} , and if so construct one, in $O(n^6)$ time.*

Proof. Apply Algorithm *LevelOne* to \mathcal{T} . For any $L' \subseteq L$, let $g(L')$ be the running time of *LevelOne*($\mathcal{T} \upharpoonright L'$). In Step 1 of the algorithm, we compute $SN(\{x_1, x_2\})$ for the n^2 pairs (x_1, x_2) in $L \times L$. By Lemma 5, Step 1 takes $O(n^5)$ time. Step 2 can be performed in $O(n^3)$ time, and Step 3 takes $\sum_{SN_i \in \mathcal{SN}} g(SN_i)$ time. Step 5 can be done in $O(n^5)$ time according to Lemma 4. In total, we have $g(L) = \sum_{SN_i \in \mathcal{SN}} g(SN_i) + O(n^5)$. Since all sets in \mathcal{SN} are disjoint, $g(L) = O(n^6)$. \square

Algorithm *LevelOne***Input:** A dense set \mathcal{T} of rooted triplets with leaf set L .**Output:** A level-1 network N consistent with \mathcal{T} , if one exists; otherwise, *null*.

```

1 for every  $x_1 \in L$  and  $x_2 \in L$  (including  $x_1 = x_2$ ) do
    Compute  $SN(\{x_1, x_2\})$ .
endfor
2 Let  $\mathcal{SN} = \{SN_1, SN_2, \dots, SN_q\}$  be the set of all maximal  $SN(\{x_1, x_2\})$ .
3 for every  $SN_i \in \mathcal{SN}$  do
    If  $|SN_i| \geq 3$  then  $N_i = \text{LevelOne}(\mathcal{T} \mid SN_i)$ ; else, let  $N_i$  be a tree distinctly
    leaf-labeled by  $SN_i$ .
endfor
4 If  $N_i$  for any  $i \in \{1, \dots, q\}$  equals null then return null.
5 If  $q = 2$  then let  $N$  be a network with a root node connected to  $N_1$  and  $N_2$ .
   Otherwise ( $q \geq 3$ ), build  $\mathcal{T}'$  from  $\mathcal{T}$ , compute  $\mathcal{N} = \text{OneHybridLeaf}(\mathcal{T}')$ , and
   check if  $\mathcal{N}$  is empty; if yes then let  $N = \text{null}$ , else select any  $N' \in \mathcal{N}$  and form
   a network  $N$  by replacing each  $\alpha_i$  in  $N'$  with  $N_i$ .
6 return  $N$ .
End LevelOne

```

Fig. 5. Constructing a level-1 phylogenetic network.

Algorithm *LevelOne* can be modified to return *all* level-1 phylogenetic networks consistent with \mathcal{T} by utilizing all the possible topologies returned by *OneHybridLeaf*. However, the running time may then become exponential since some inputs are consistent with an exponential number of different level-1 networks. (At each recursion level, although the partition of the leaves into \mathcal{SN} is unique when the input is dense, there may be more than one way to merge the recursively computed subnetworks for the SN -sets into a valid network.)

5 Conclusion

This paper presents a polynomial-time algorithm for inferring a level-1 phylogenetic network from a dense set of rooted triplets. This problem is not only interesting from a combinatorial point of view, but also biologically sound since rooted triplets can be obtained accurately by using maximum likelihood or directly through experiments. In the future, we plan to further improve the time complexity of our main algorithm and to investigate the computational complexity of the problem when \mathcal{T} is not dense. Also, we would like to know if it is possible to construct a level- f phylogenetic network from a dense set of rooted triplets in polynomial time for any constant $f > 1$.

References

1. A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.

2. D. Bryant. *Building Trees, Hunting for Trees, and Comparing Trees: Theory and Methods in Phylogenetic Analysis*. PhD thesis, University of Canterbury, Christchurch, New Zealand, 1997.
3. B. Chor, M. Hendy, and D. Penny. Analytic solutions for three-taxon ML_{MC} trees with variable rates across sites. In *Proc. of the 1st Workshop on Algorithms in Bioinformatics* (WABI 2001), volume 2149 of *LNCS*, pages 204–213. Springer, 2001.
4. C. Choy, J. Jansson, K. Sadakane, and W.-K. Sung. Computing the maximum agreement of phylogenetic networks. In *Proc. of Computing: the 10th Australasian Theory Symposium* (CATS 2004), pages 33–45. Elsevier, 2004.
5. T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, Massachusetts, 1990.
6. L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin. Inferring ordered trees from local constraints. In *Proc. of Computing: the 4th Australasian Theory Symposium* (CATS'98), volume 20(3) of *Australian Computer Science Communications*, pages 67–76. Springer-Verlag Singapore, 1998.
7. L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin. On the complexity of constructing evolutionary trees. *Journal of Combinatorial Optimization*, 3:183–197, 1999.
8. D. Gusfield, S. Eddhu, and C. Langley. Efficient reconstruction of phylogenetic networks with constrained recombination. In *Proc. of the Computational Systems Bioinformatics Conference* (CSB2003), pages 363–374, 2003.
9. J. Hein. Reconstructing evolution of sequences subject to recombination using parsimony. *Mathematical Biosciences*, 98(2):185–200, 1990.
10. M. R. Henzinger, V. King, and T. Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Algorithmica*, 24(1):1–13, 1999.
11. J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM*, 48(4):723–760, 2001.
12. J. Jansson. On the complexity of inferring rooted evolutionary trees. In *Proc. of the Brazilian Symp. on Graphs, Algorithms, and Combinatorics* (GRACO'01), volume 7 of *Electronic Notes in Discrete Mathematics*, pages 121–125. Elsevier, 2001.
13. J. Jansson, J. H.-K. Ng, K. Sadakane, and W.-K. Sung. Rooted maximum agreement supertrees. In *Proc. of Latin American Theoretical Informatics* (LATIN 2004), volume 2976 of *LNCS*, pages 499–508, 2004.
14. T. Jiang, P. Kearney, and M. Li. A polynomial time approximation scheme for inferring evolutionary trees from quartet topologies and its application. *SIAM Journal on Computing*, 30(6):1942–1961, 2001.
15. S. Kannan, E. Lawler, and T. Warnow. Determining the evolutionary tree using experiments. *Journal of Algorithms*, 21(1):26–50, 1996.
16. P. Kearney. Phylogenetics and the quartet method. In T. Jiang, Y. Xu, and M. Q. Zhang, editors, *Current Topics in Computational Molecular Biology*, pages 111–133. The MIT Press, Massachusetts, 2002.
17. L. Nakhleh, T. Warnow, and C. R. Linder. Reconstructing reticulate evolution in species – theory and practice. In *Proc. of the 8th Annual International Conference on Research in Computational Molecular Biology* (RECOMB 2004), to appear.
18. D. Posada and K. A. Crandall. Intraspecific gene genealogies: trees grafting into networks. *TRENDS in Ecology & Evolution*, 16(1):37–45, 2001.
19. M. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9(1):91–116, 1992.
20. L. Wang, K. Zhang, and L. Zhang. Perfect phylogenetic networks with recombination. *Journal of Computational Biology*, 8(1):69–78, 2001.

Author Index

- Adamy, Udo 62
Ahn, Hee-Kap 259
Akutsu, Tatsuya 249
Arge, Lars 1
Ausiello, Giorgio 290
Azuma, Machiko 309
- Baille, Fabien 219
Bampis, Evripidis 219
Brass, Peter 259
- Chan, Wun-Tat 210
Chandran, L. Sunil 151
Chen, Danny Z. 112, 238
Chen, Zhi-Zhong 319, 450
Cheong, Otfried 259
Chin, Francis Y.L. 53
Chun, Jinhee 238
Cicalese, Ferdinando 82
- Deĭneko, Vladimir G. 268
Demange, Marc 290
Deppe, Christian 82
Du, Ying 112
- Hoffmann, Michael 62, 268
- Imamura, Tomokazu 132
Iwama, Kazuo 132
- Jansson, Jesper 462
Jonsson, Peter 370
- Kaminski, Michael 171
Kato, Naoki 238
Kim, Hee-Chul 412
Kim, Jeong Han 2
Kim, Jin Wook 440
Koizumi, Koichi 92
- Laforest, Christian 219
Lam, Tak-Wah 210
Laura, Luigi 290
Lefmann, Hanno 43
Leong, Hon Wai 339
- Leung, Ho-Fung 432
Li, Guojun 229
Li, Shuai Cheng 339
Li, Shuguang 229
Lim, Andrew 102, 122, 349
Lim, Hyeong-Seok 412
- Mahajan, Meena 33
Miao, Zhaowei 122
Miura, Kazuyuki 309
Mizuki, Takaaki 92
Moczurad, Małgorzata 72
Moczurad, Włodzimierz 72
Mundici, Daniele 82
- Na, Hyeon-Suk 259
Nakanishi, Masaki 179
Nakata, Toshio 422
Narayanaswamy, N.S. 151, 329
Nishizeki, Takao 92, 309
Nomura, Kumiko 300
Nordh, Gustav 370
- Okamoto, Yoshio 268
Onishi, Kensuke 13
- Park, Jung-Heum 392, 412
Park, Kunsoo 440
Park, Kyoung-Wook 412
Paschos, Vangelis 290
Peng, Zeshan 432
- Qian, Jianbo 53
Quek, Steven K. 339
- Rama, Raghavan 33
Ratsaby, Joel 198
Rettinger, Robert 360
Rodrigues, Brian 122, 349
- Shin, Chan-Su 259
Solymosi, József 62
Stojaković, Miloš 62
Sugihara, Kokichi 3
Sung, Wing-Kin 462

- Takahashi, Haruhisa 161
Takaoka, Tadao 278
Tan, Tony 171
Tanaka, Akira 161
Tayu, Satoshi 300
Ting, Hing-Fung 210, 432
Tokuyama, Takeshi 238
Tomita, Etsuji 161
Tsukiji, Tatsue 132, 450

Ueno, Shuichi 300

Vigneron, Antoine 259
Vijayakumar, S. 33
Vikas, Narayan 380
Vinodchandran, N. Variyam 188

Wang, Cao An 53
Wang, Fan 102, 349
Woeginger, Gerhard J. 268
Woelfel, Philipp 23
Wong, Prudence W.H. 210
Wu, Xiaodong 112

Xu, Zhou 102, 122, 349

Yamada, Toshinori 402

Zhang, Shaoqiang 229
Zhang, Yong 143
Zheng, Xizhong 360
Zhu, Hong 143